Development of the ROOT system History & Perspectives

Brookhaven National Lab

August 11th, 2008

René Brun, Philippe Canal

Outline

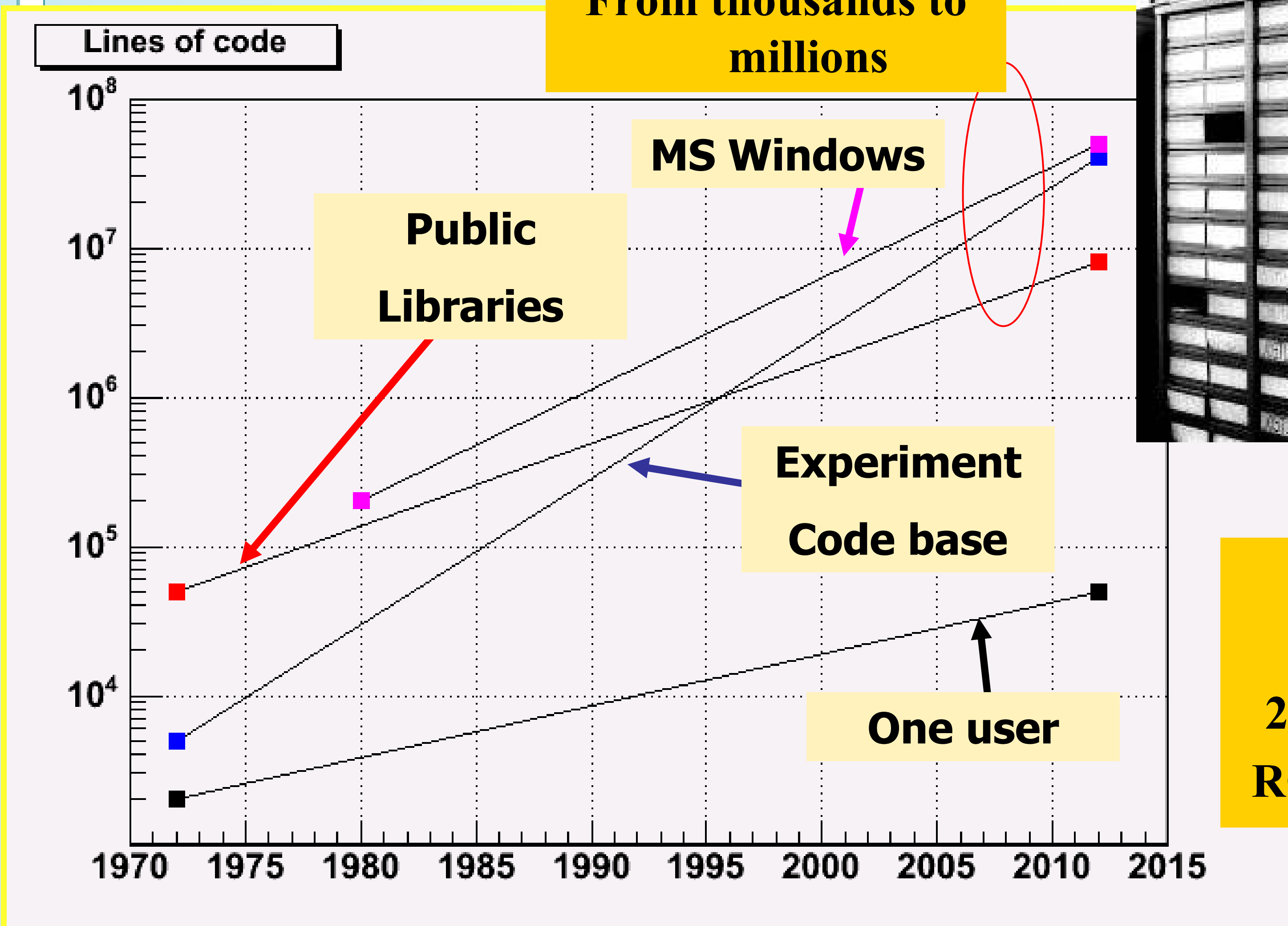
- Comments On The Evolution of Computing
- Challenges Ahead And ROOT's Take on Them
- History of ROOT I/O

Fantastic Evolution of Computing

- Processors: x2000
- Memory: x1000
- Storage: x5000
- Networks : x100000

In 30 years only

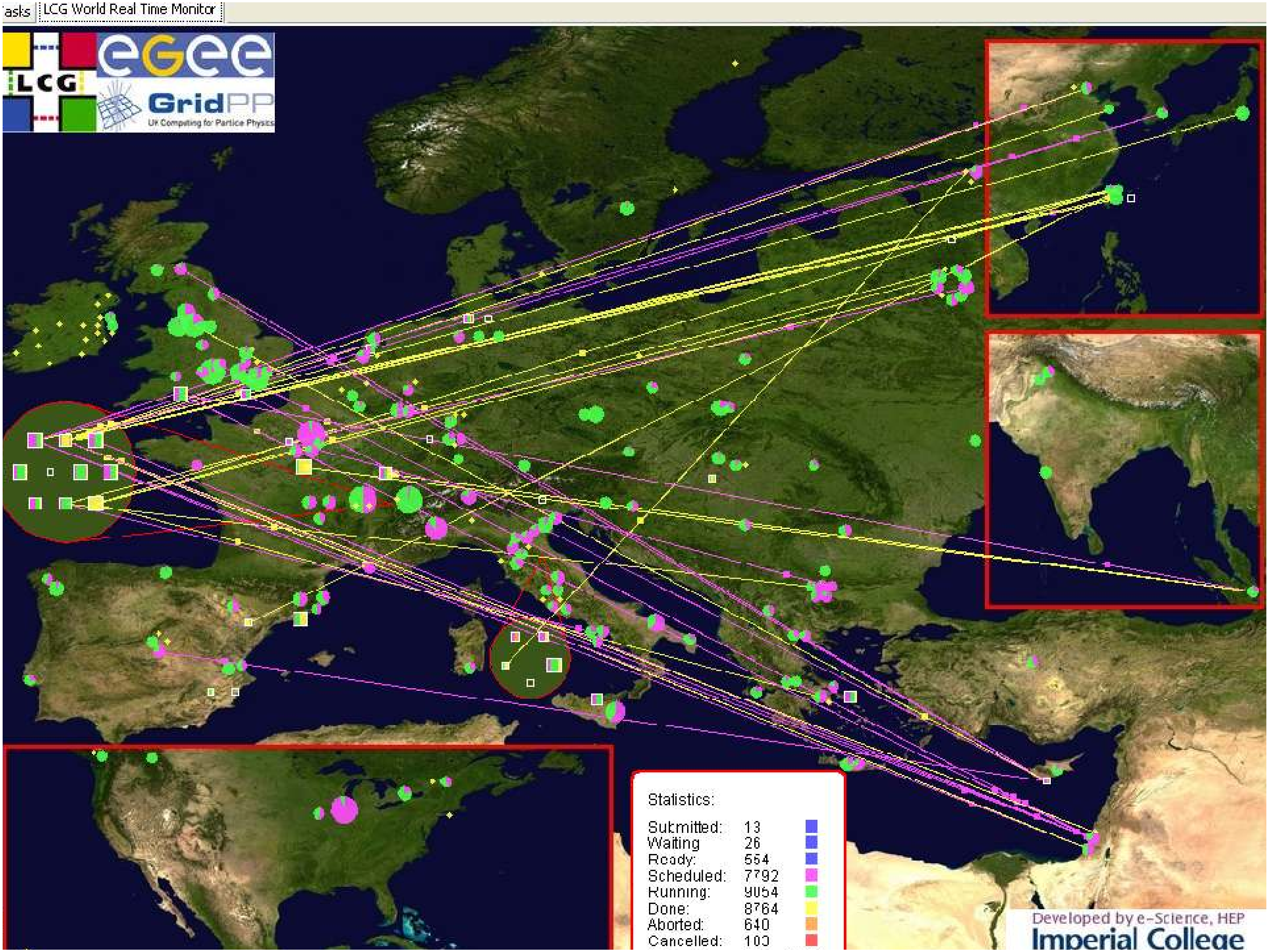
Program size
(lines of code)



2000 cards per box

24 boxes per rack

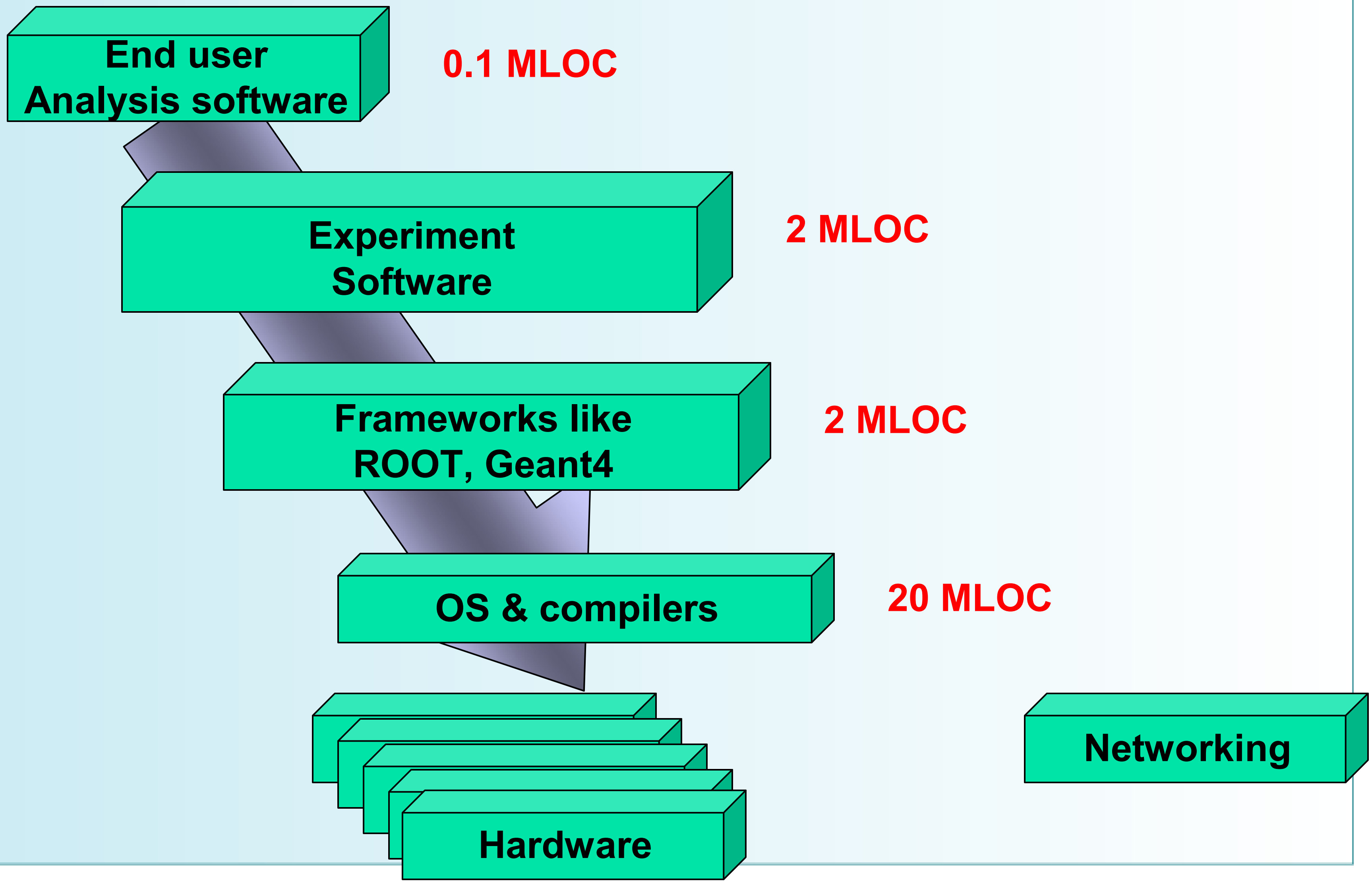
ROOT = 100 racks

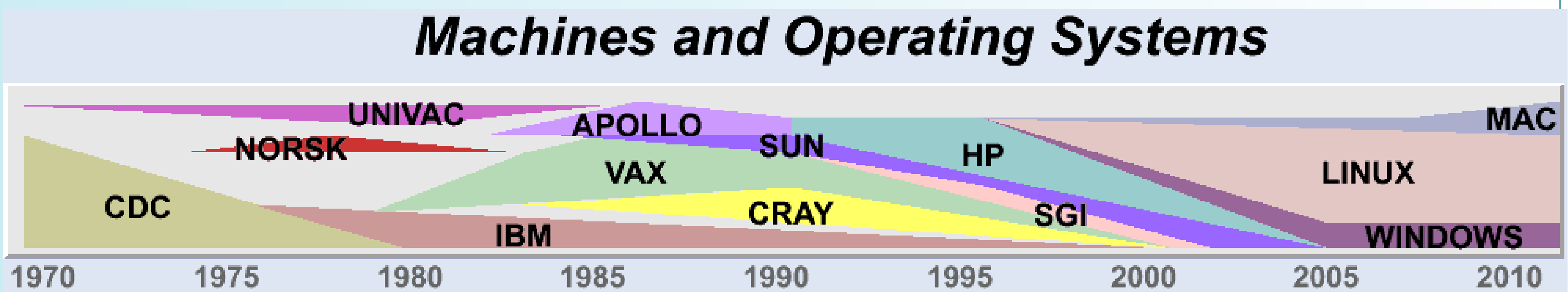

asks
LCG World Real Time Monitor
LCG
eGee
GridPP
UK Computing for Particle Physics
Statistics:
Submitted: 13
Waiting 26
Ready: 554
Scheduled: 7792
Running: 9054
Done: 8764
Aborted: 640
Cancelled: 100
Developed by e-Science, HEP
Imperial College

The main general software packages

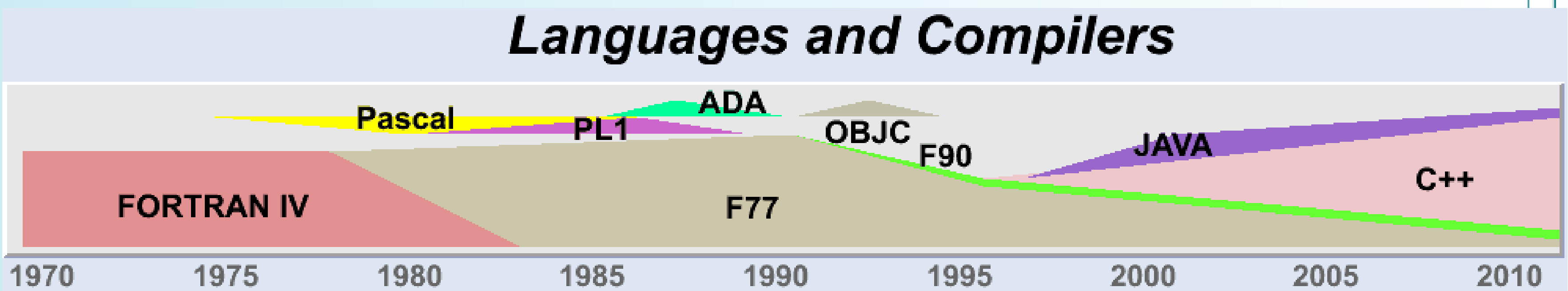
- **1965: each physicist writes his/her own analysis program**
- **1975: First tools:**
  - **Histograms/statistics(HBOOK), Visualisation (GD3),**
  - **Minimisation(Minuit), Simulation (GEANT1,2).**
- **1985:**
  - **Super Minis (VAX) and workstations (Apollo, VAX,IBM).**
  - **A big step for detector simulation (GEANT3),**
  - **A big step for interactive analysis (PAW).**
- **1995:**
  - **PAW, GEANT3 stables**
  - **Investigation of Object-Oriented systems**
  - **Failure of commercial products (Objectivity, Iris Explorer,..)**
  - **The challenger ROOT**
- **2005: ROOT, GEANT4**

Software Hierarchy

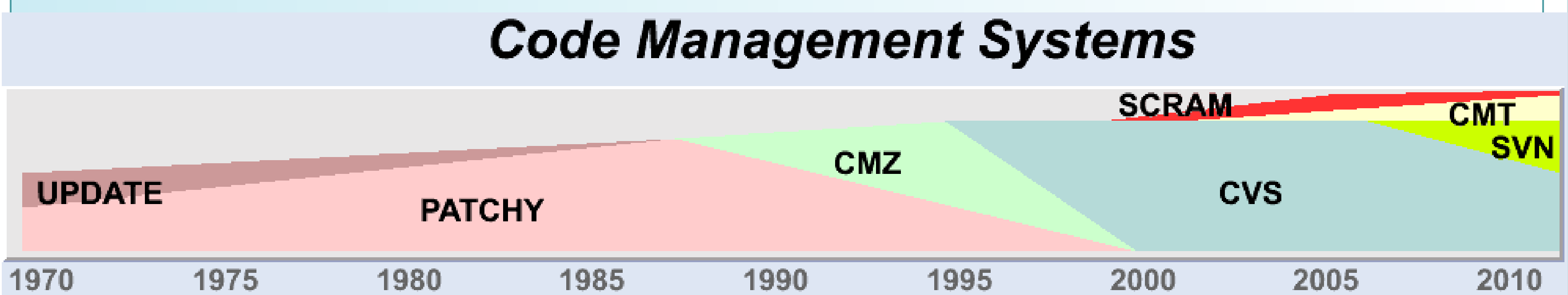




Machines and Operating Systems
UNIVAC
NORSK
CDC
APOLLO
VAX
SUN
HP
IBM
CRAY
SGI
MAC
LINUX
WINDOWS
1970
1975
1980
1985
1990
1995
2000
2005
2010

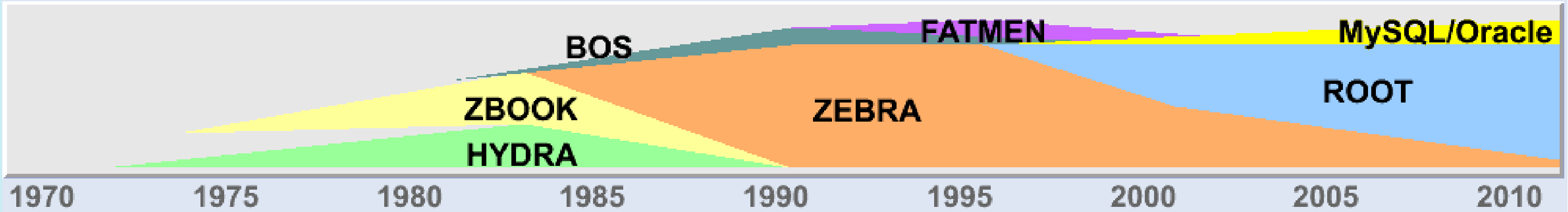


Languages and Compilers
Pascal
PL1
ADA
OBJC
F90
JAVA
C++
FORTRAN IV
F77
1970
1975
1980
1985
1990
1995
2000
2005
2010

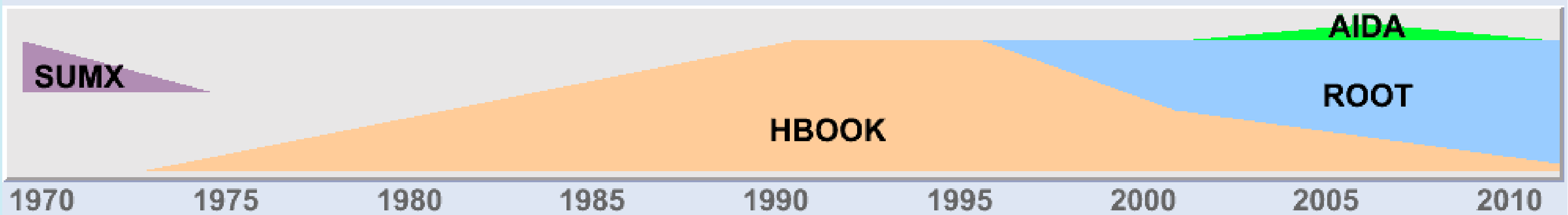


Code Management Systems
SCRAM
CMT
SVN
CMZ
UPDATE
PATCHY
CVS
1970
1975
1980
1985
1990
1995
2000
2005
2010

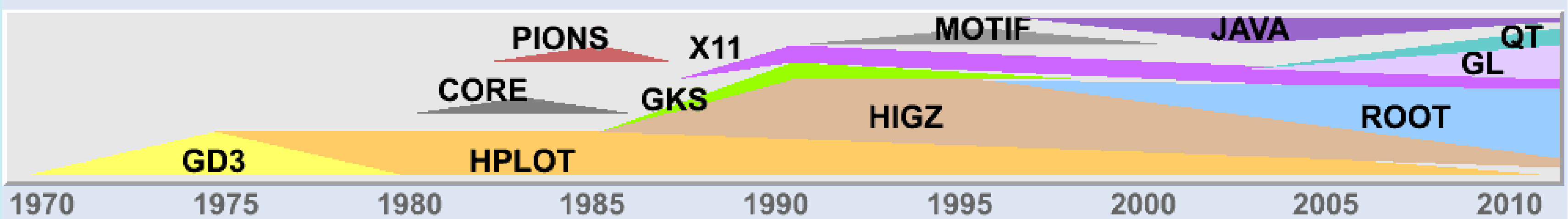
Data Structures Management Systems



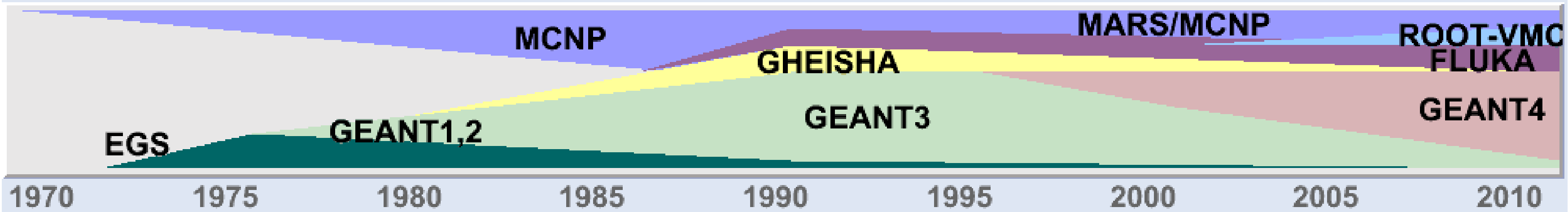
Histograms Systems



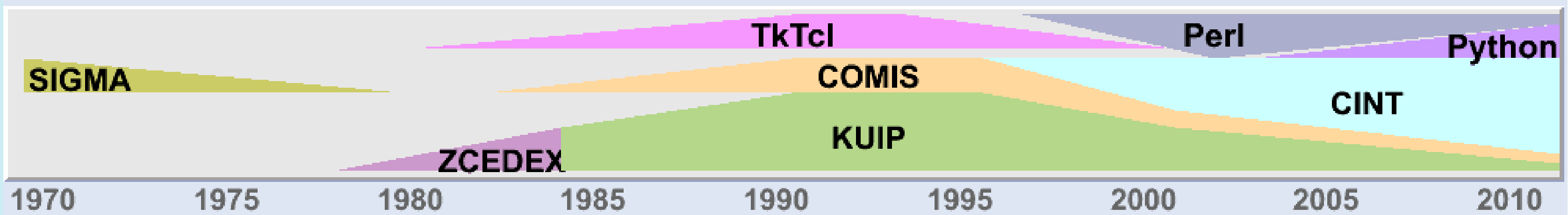
Graphics and GUIs



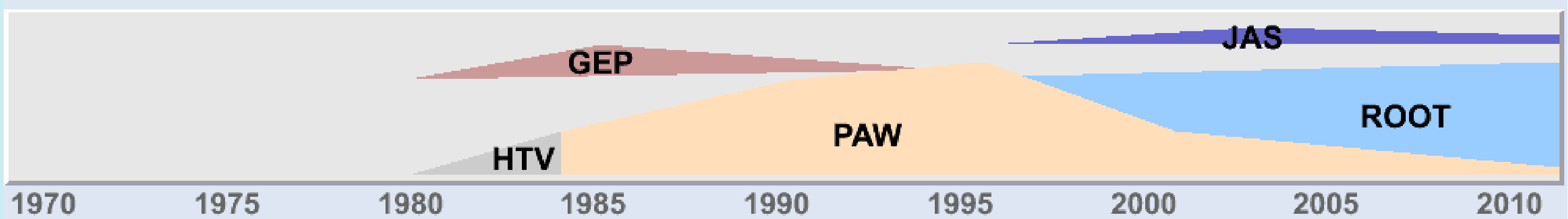
Detector Simulation Systems



Scripting Systems and Interpreters



Interactive Data Analysis Systems



The crystal ball in 1988

- Fortran90 seems the obvious way to go
- OSI protocols to replace TCP/IP
- Processors: Vector or MPP machines
- PAW,Geant3,Bos,Zebra: Adapt them to F90X
- Methodoly trend: Entity Relationship Model
- Parallelism: vectorization or MPP (SIMD and MIMD)

- BUT hard to anticipate that
  - The WEB will come less than 3 years later
  - The 1993/1994 revolution for languages and projects
  - The rapid grow in CPU power starting in 1994 (Pentium)

Situation in 1998

- LHC projects moving to C++
- Several projects proposing to use Java
- Huge effort with OODBMS (ie Objectivity)
- Investigate Commercial tools for data analysis
- ROOT development not encouraged
- Vast majority of users very sceptic.

- RAM <256 MB
- Program Size < 32 MB
- <500 KLOcs
- libs < 10
- static linking
- HSM: tape->Disk pool <1 TByte
- Network 2MB/s

The crystal ball in 1998

- C++ now, Java in 2000
- Future is OODBMS (ie Objectivity)
- Central Event store accessed through the net
- Commercial tools for data analysis

- But fortunately a few people did not believe in this direction :☺
- First signs of problems with Babar
- FNAL RUN2 votes for ROOT in 1998
- GRID: an unknown word in 1997 :☺

Situation in 2008

- It took far more time than expected to move people to C++ and the new frameworks.
- ROOT de facto standard for I/O and interactive analysis.
- The GRID:
- Experiment frameworks are monsters

The challenges

- **Simplify the use of software systems**
- **Granularity**
- **Hope to see self-descriptive languages**
- **Interpreters + compilers**
- **Importance of caches on LAN and WAN**
- **« Task » oriented programming**
- **GUI with dynamic configuration**
- **Everything from the browser ?**
- **Graphics based on GL: Post X11 et QT**
- **Execute anywhere from anywhere**
- **Evolution of the execution (main --> plug-ins)**
- **Hardware force parallelism**
- **Extension of client-server models**
- **Data analysis**
- **-from batch to interactive systems**
- **-from sequential processing to parallelism**

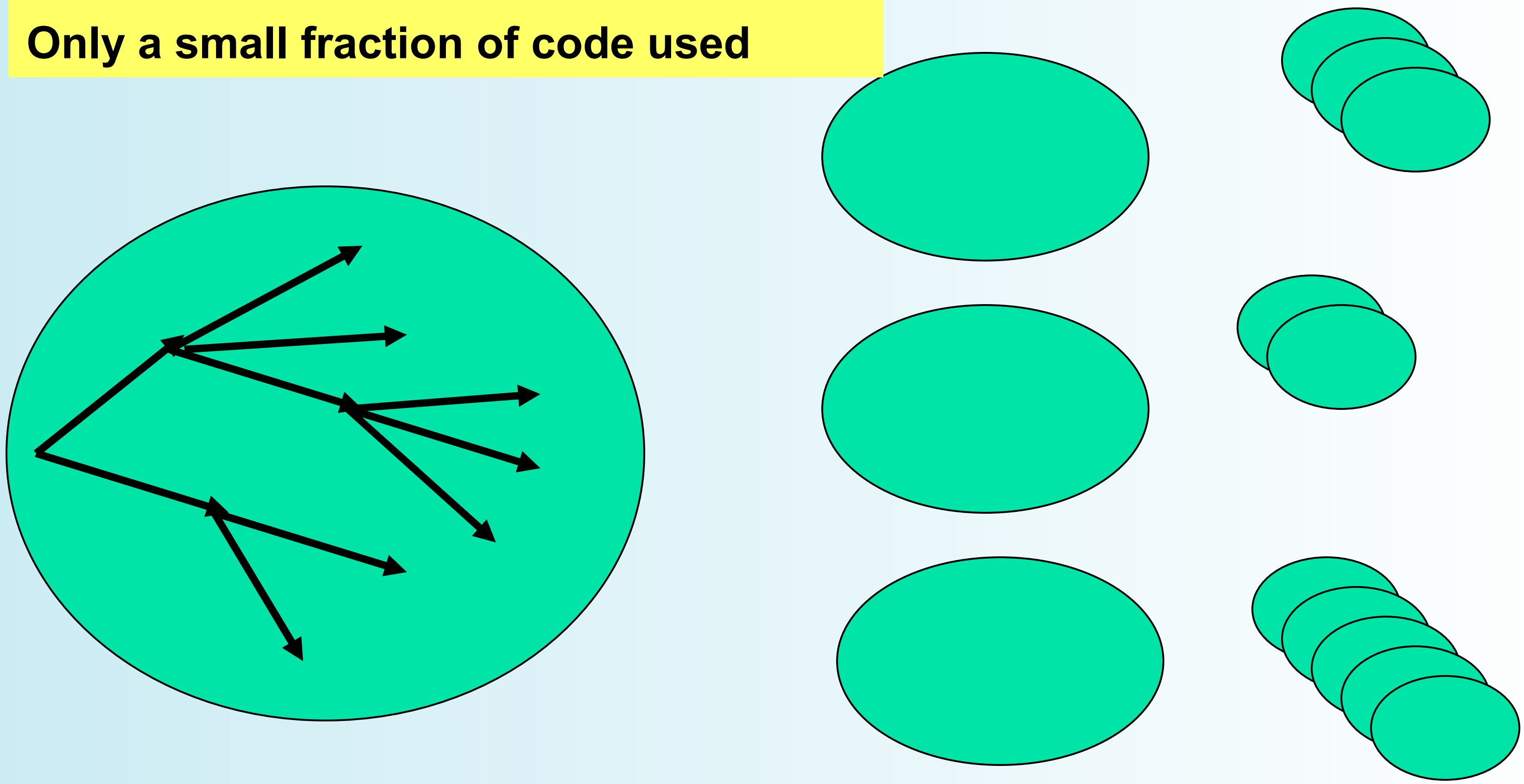
Challenge
Usability: Making things SIMPLER

- Guru view vs user view
- A normal user has to learn too many things before being able to do something useful.
- LHC frameworks becoming monsters
- fighting to work on 64 bits with <2 GBytes
- Executable take for ever to start because too much code linked (shared libs with too many dependencies)
- fat classes vs too many classes
- It takes time to restructure large systems to take advantage of plug-in managers.

Challenge ++
Problem decomposition

Will have to deal with many shared libs

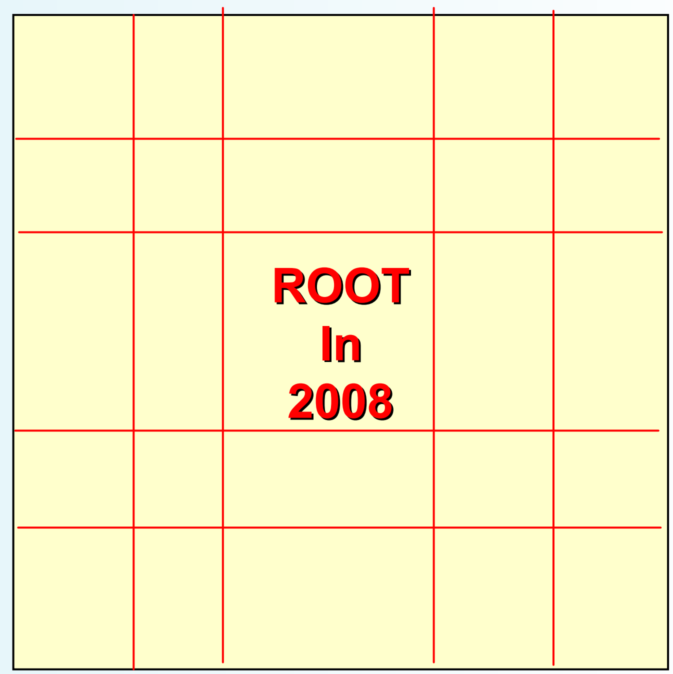
Only a small fraction of code used



Some Facts

100 shared libs

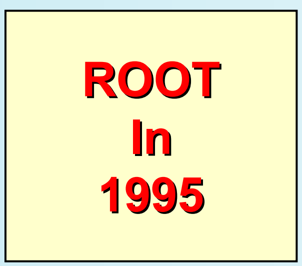
2000 classes



Plug-in manager

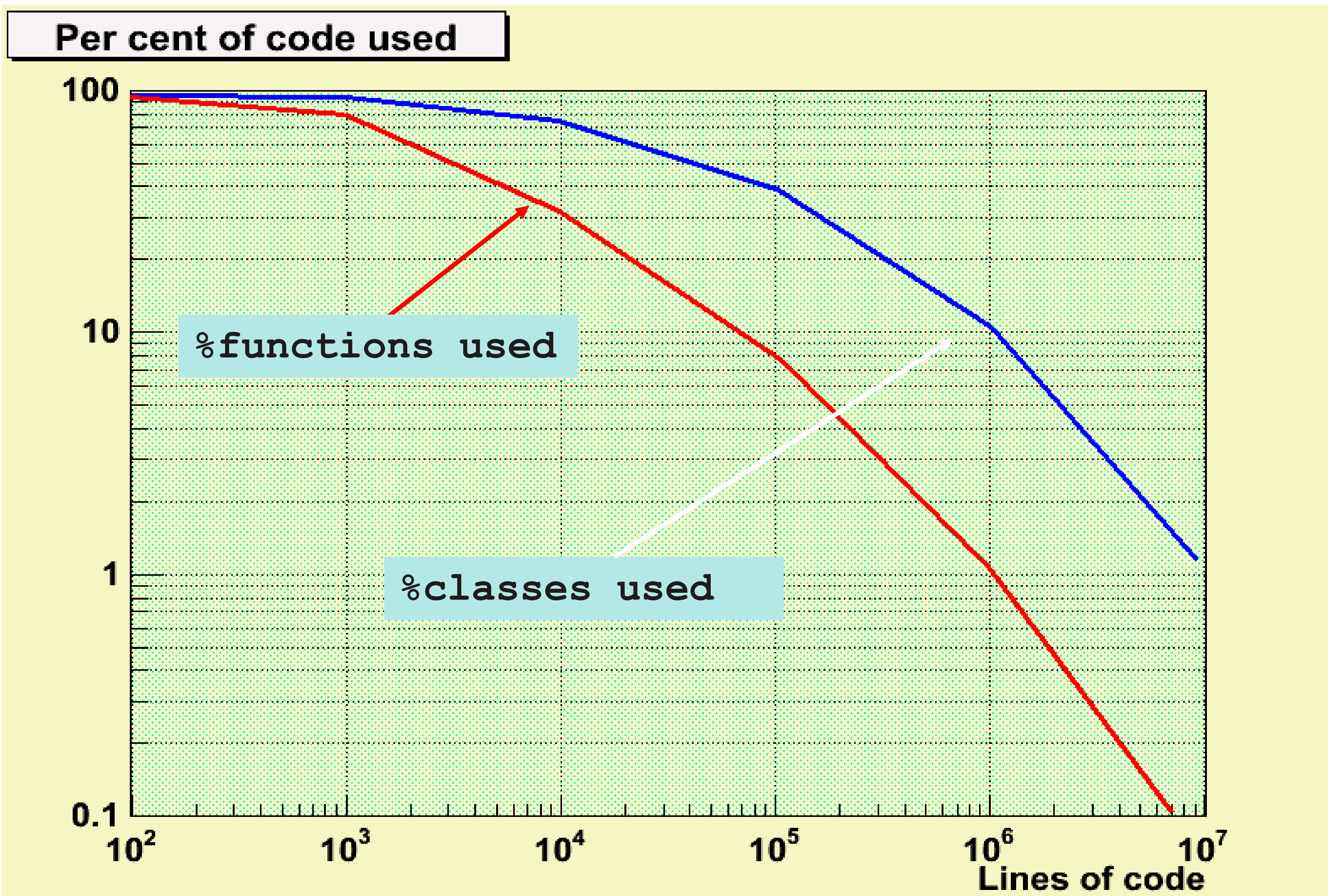
10 shared libs

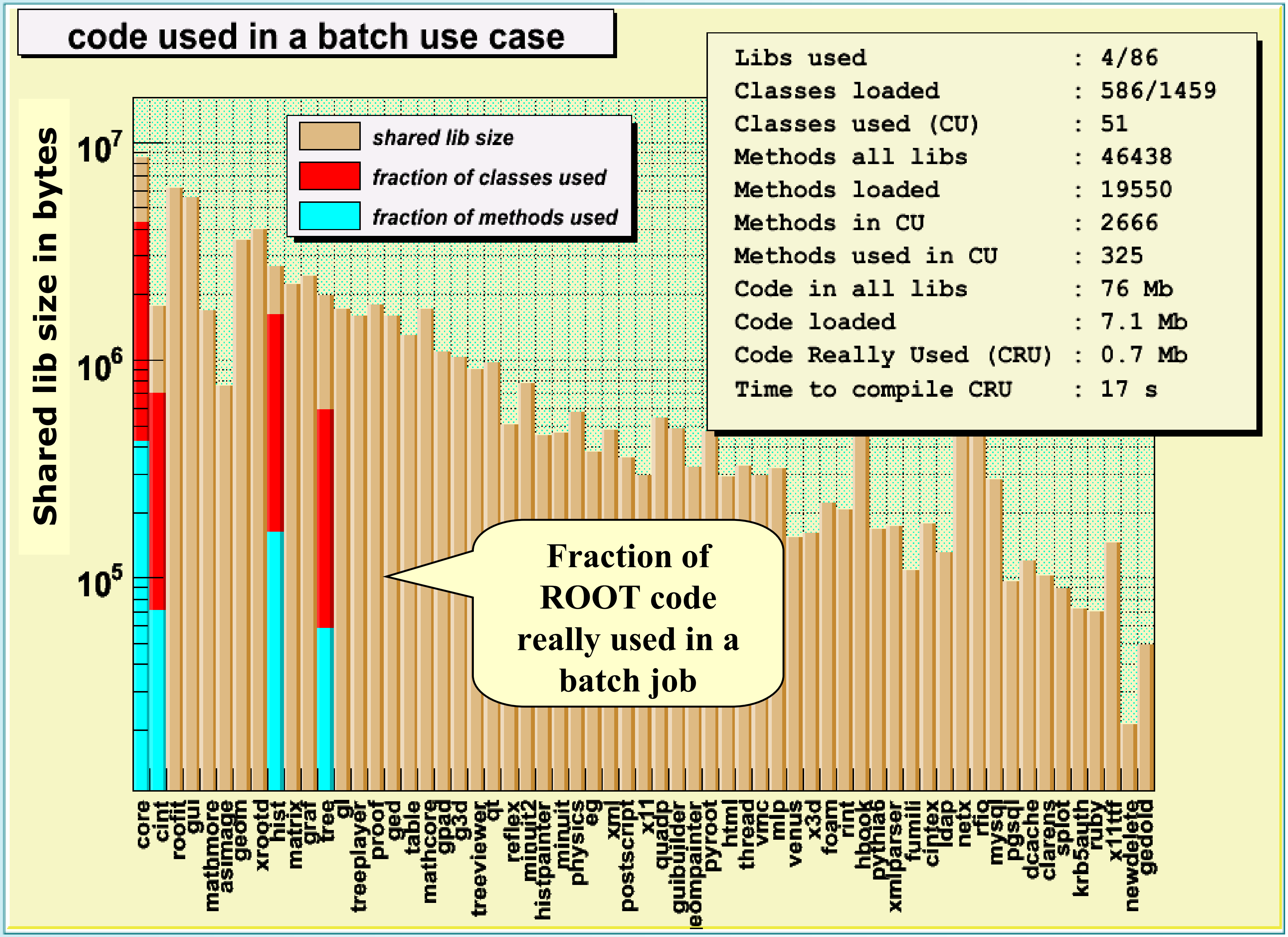
200 classes



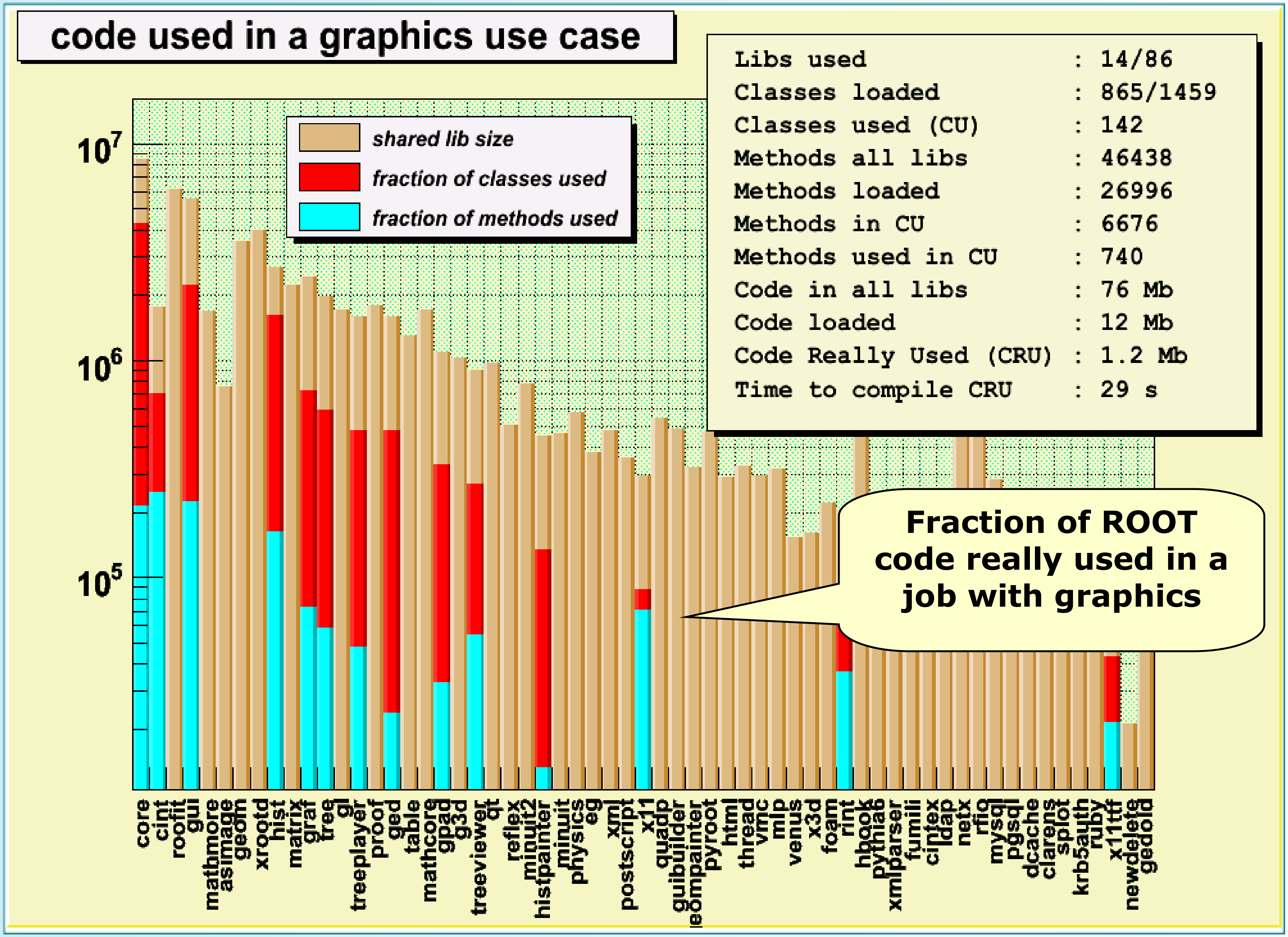
PAW model

Fraction of code really used in one program



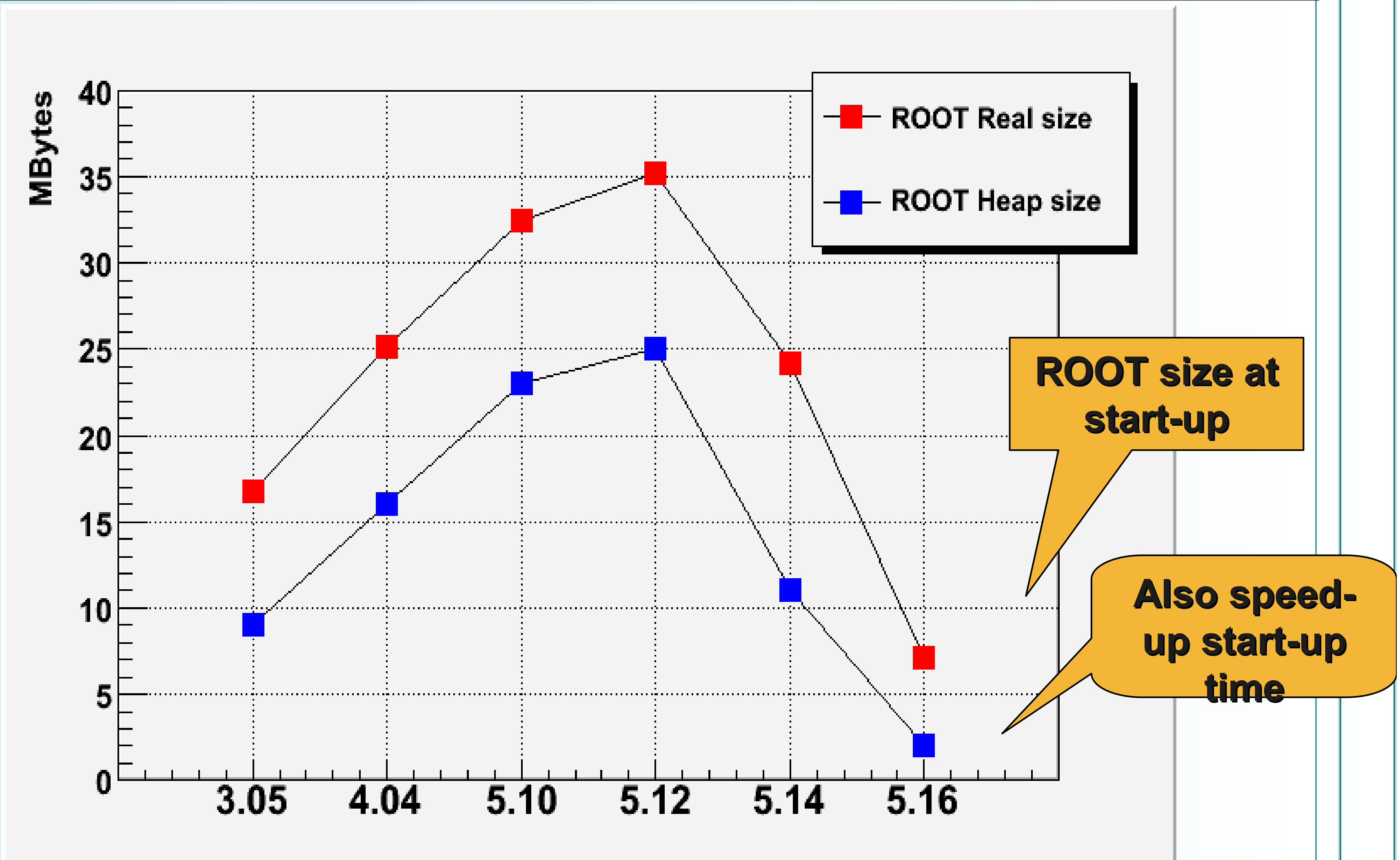


code used in a batch use case
Shared lib size in bytes
10^7
10^6
10^5
shared lib size
fraction of classes used
fraction of methods used
Libs used : 4/86
Classes loaded : 586/1459
Classes used (CU) : 51
Methods all libs : 46438
Methods loaded : 19550
Methods in CU : 2666
Methods used in CU : 325
Code in all libs : 76 Mb
Code loaded : 7.1 Mb
Code Really Used (CRU) : 0.7 Mb
Time to compile CRU : 17 s
Fraction of ROOT code really used in a batch job
core cint roofit gui mathmore asimage geom xrootd hist matrix graf tree gl treeplayer proof ged table mathcore gpad g3d treeviewer qt reflex minuit2 histpainter minuit physics eg xml postscript x11 quadp guibuilder eompainter pyroot html thread vmc mlp venus x3d foam rint hbook pythia6 xmlparser fumili cintex ldap netx rfio mysql pgsql dcache clarens splot krb5auth ruby x11ttf newdelete gedold



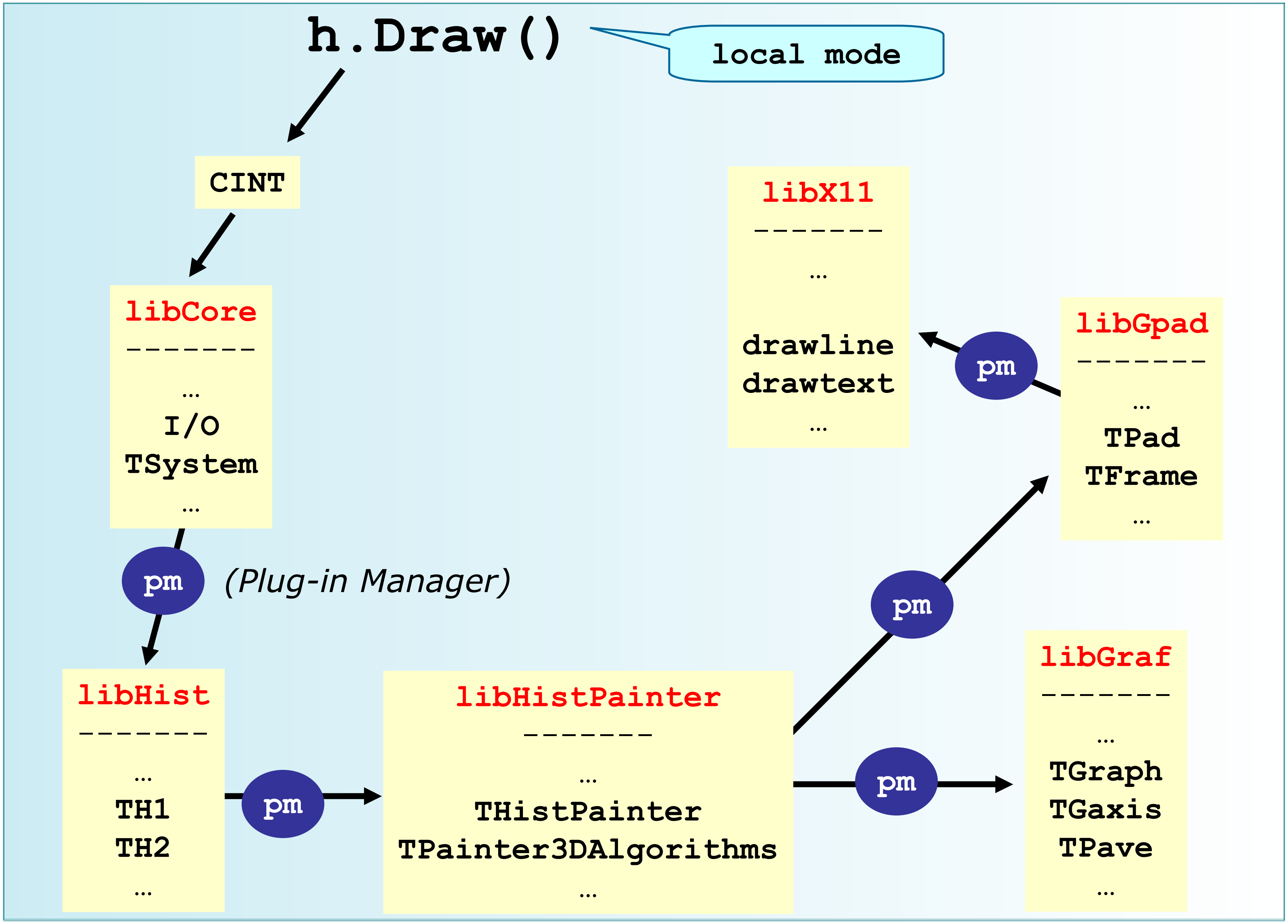
code used in a graphics use case
shared lib size
fraction of classes used
fraction of methods used
10^7
10^6
10^5
Libs used : 14/86
Classes loaded : 865/1459
Classes used (CU) : 142
Methods all libs : 46438
Methods loaded : 26996
Methods in CU : 6676
Methods used in CU : 740
Code in all libs : 76 Mb
Code loaded : 12 Mb
Code Really Used (CRU) : 1.2 Mb
Time to compile CRU : 29 s
Fraction of ROOT code really used in a job with graphics
core cint roofit gui mathmore asimage geom xrootd hist matrix graf tree gl treeplayer proof ged table mathcore gpad g3d treeviewer qt reflex minuit2 histpainter minuit physics eg xml postscript x11 quadp guibuilder geompainter pyroot html thread vmc mlp venus x3d foam rint hbook pythia6 xmlparser fumili cintex ldap netx rfio mysql pgsql dcache clarens splot krb5auth ruby x11ttf newdelete gedold

Large Heap Size Reduction



Challenge ++
Sophisticated Plug-in Managers

- When using a large software base distributed with hundred of shared libs, it is essential to discover automatically where to find a class.
- The interpreters must be able to auto-load the corresponding libraries


h.Draw()
local mode
CINT
libCore

…
I/O
TSystem
…
pm
(Plug-in Manager)
libHist

…
TH1
TH2
…
pm
libHistPainter

…
THistPainter
TPainter3DAlgorithms
…
pm
libGpad

…
TPad
TFrame
…
pm
libX11

…
drawline
drawtext
…
pm
libGraf

…
TGraph
TGaxis
TPave
…

Challenge ++
Languages

- C++ clear winner in our field and also other fields
- see, eg a recent compilation at http://www.lextrait.com/vincent/implementations.html
- From simple C++ to complex templated code
- Unlike Java, no reflexion system. This is essential for I/O and interpreters.
- C++2009: better thread support, Aspect-oriented
- C++2014: first reflexion system?

Challenge ++
Opportunistic Use of Interpreters

- Use interpreted code only for:
  - External and thin layer (task organizer)
  - Slots execution in GUI signal/slots
  - Dynamic GUI builder in programs like event displays.
- Instead optimize the compiler/linker interface (eg **ACLiC**) to have
  - Very fast compilation/linking when performance is not an issue
  - Slower compilation but faster execution for the key algorithms
- ie use ONE single language for 99% of your code and the interpreter of your choice for the layer between shell programming and program orchestration.

Interpreter & Compiler integration

```
root > .x script.C
root >  DoSomething(...);
root > .x script.C++
root > .x script.C+
```

execute file **script.C**

execute function **DoSomething**

compile file **script.C**
and execute it

compile file **script.C**
if file has been modified.
execute it

```
gROOT->ProcessLine(".L script.C+");
gROOT->ProcessLine("DoSomething(...)");
```

same from
compiled
or interpreted
code

Challenge ++
The Language Reflexion System

- Develop a robust dictionary system that can be migrated smoothly to the reflexion system to be introduced in C++ in a few years.
- Meanwhile reduce the size of dictionaries by doing more things at run time.
- Replace generated code by objects stored in ROOT files.
- Direct calls to compiled code from the interpreter instead of function stubs. This is compiler dependent (mangling/de-mangling symbols).

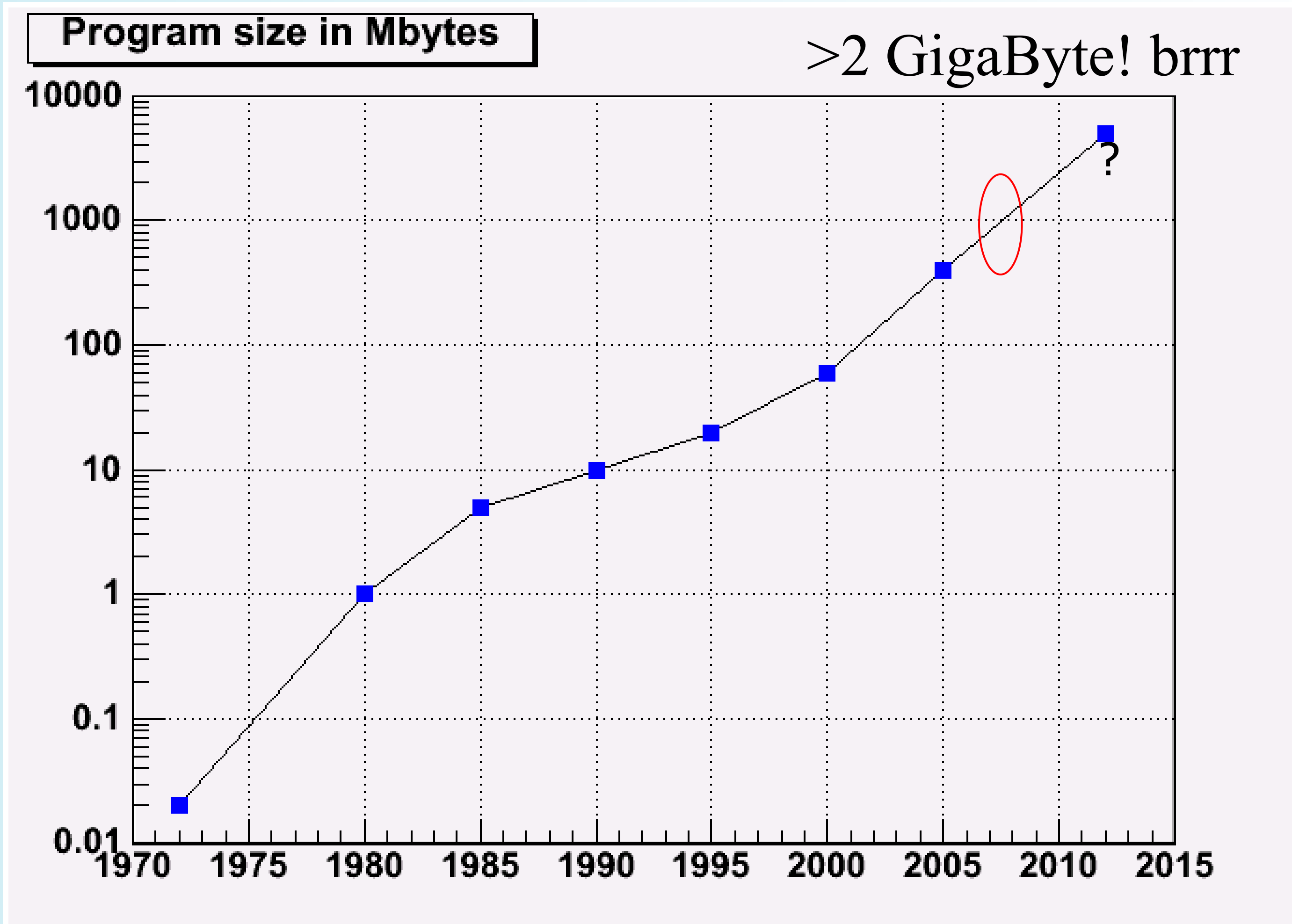
Challenge ++
Code Performance

- HEP code does not exploit hardware (see S.Jarp talk at CHEP07)
- Large data structures spread over >100 Megabytes
- templated code pitfall
  - STL code duplication
  - good perf improvement when testing with a toy.
  - disaster when running real programs.
- std::string passed by value
- abuse of new/delete for small objects or stack objects
- linear searches vs hash tables or binary search
- abuse of inheritance hierarchy
- code with no vectors -> do not use the pipeline

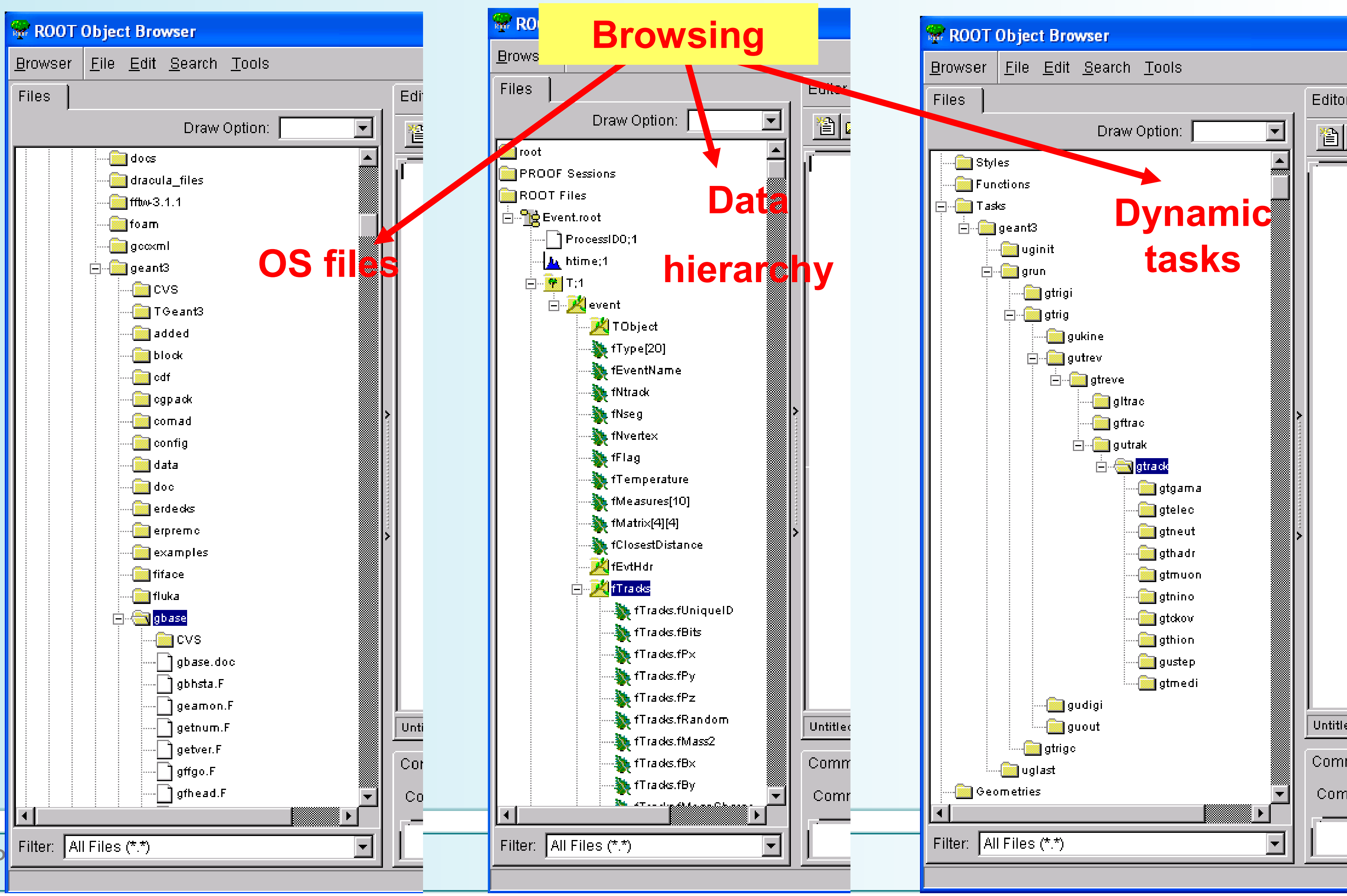
Challenge ++
Software Correctness

- big concern with multi million lines of code
- validation suite
- unit test
- combinatorial test
- nightly builds (code + validation suite)

Programs Size (RAM)



Challenge ++
Towards Task-oriented programming

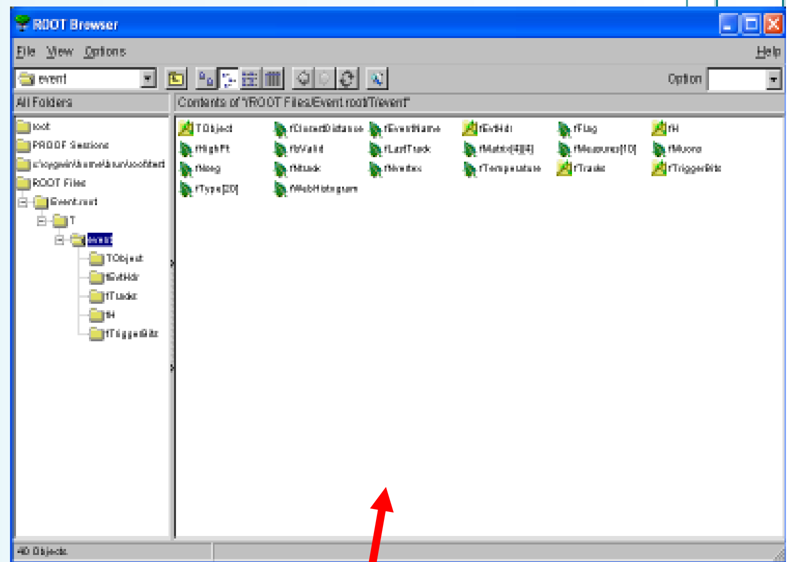


Challenge ++
Customizable and Dynamic GUIs

- From a standard browser (eg ROOT TBrowser) on must be able to include user-defined GUIs.
- The GUIs should not require any pre-processor.
- They can be executed/loaded/modified in the same session.

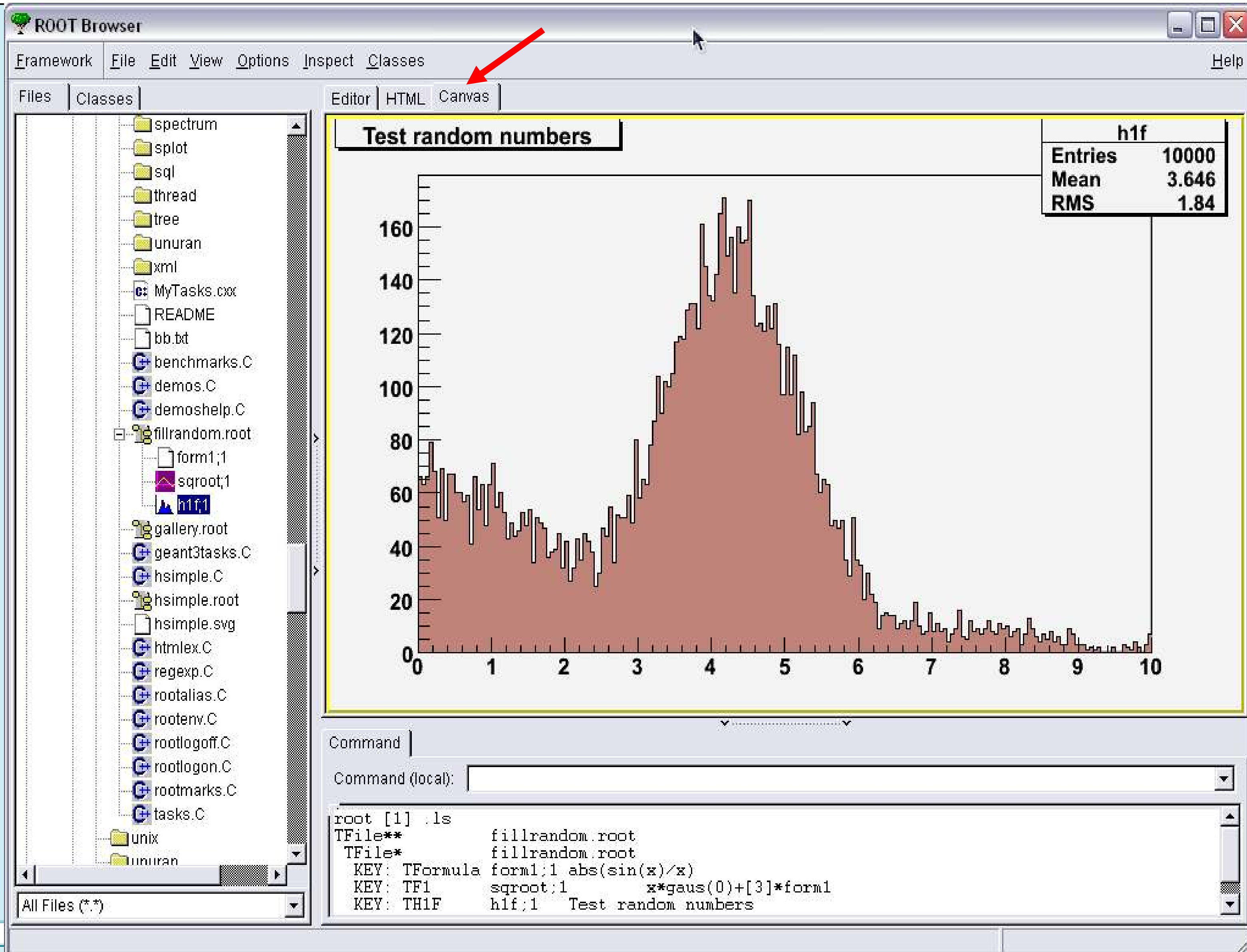
Browser Improvements

- The browser (TBrowser and derivatives) is an essential component (from beginners to advanced applications).
- It is currently restricted to the browsing of ROOT files or Trees.
- We are extending TBrowser such that it could be the central interface and the manager for any GUI application (editors, web browsers, event displays, etc).

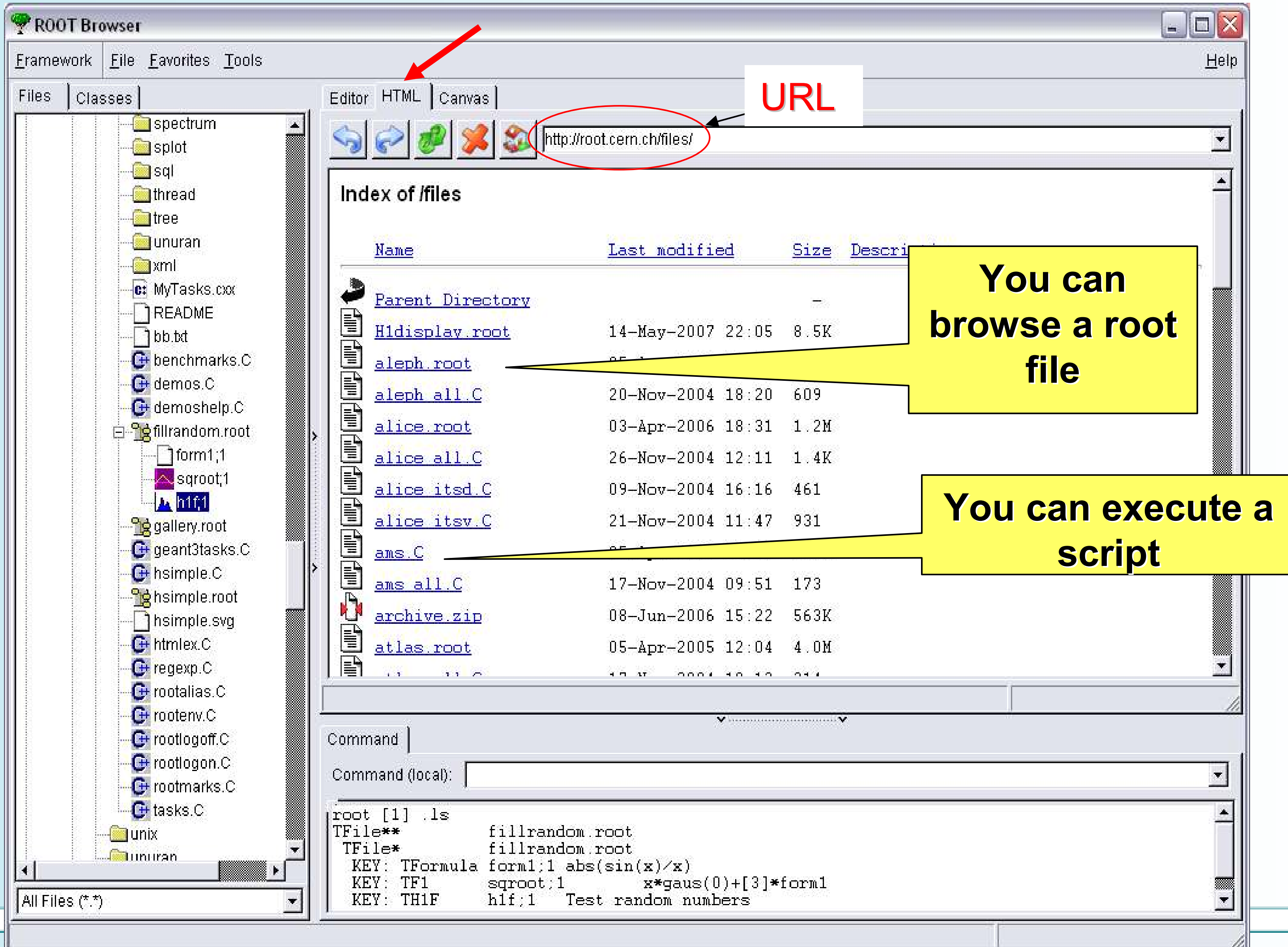


Old/current browser

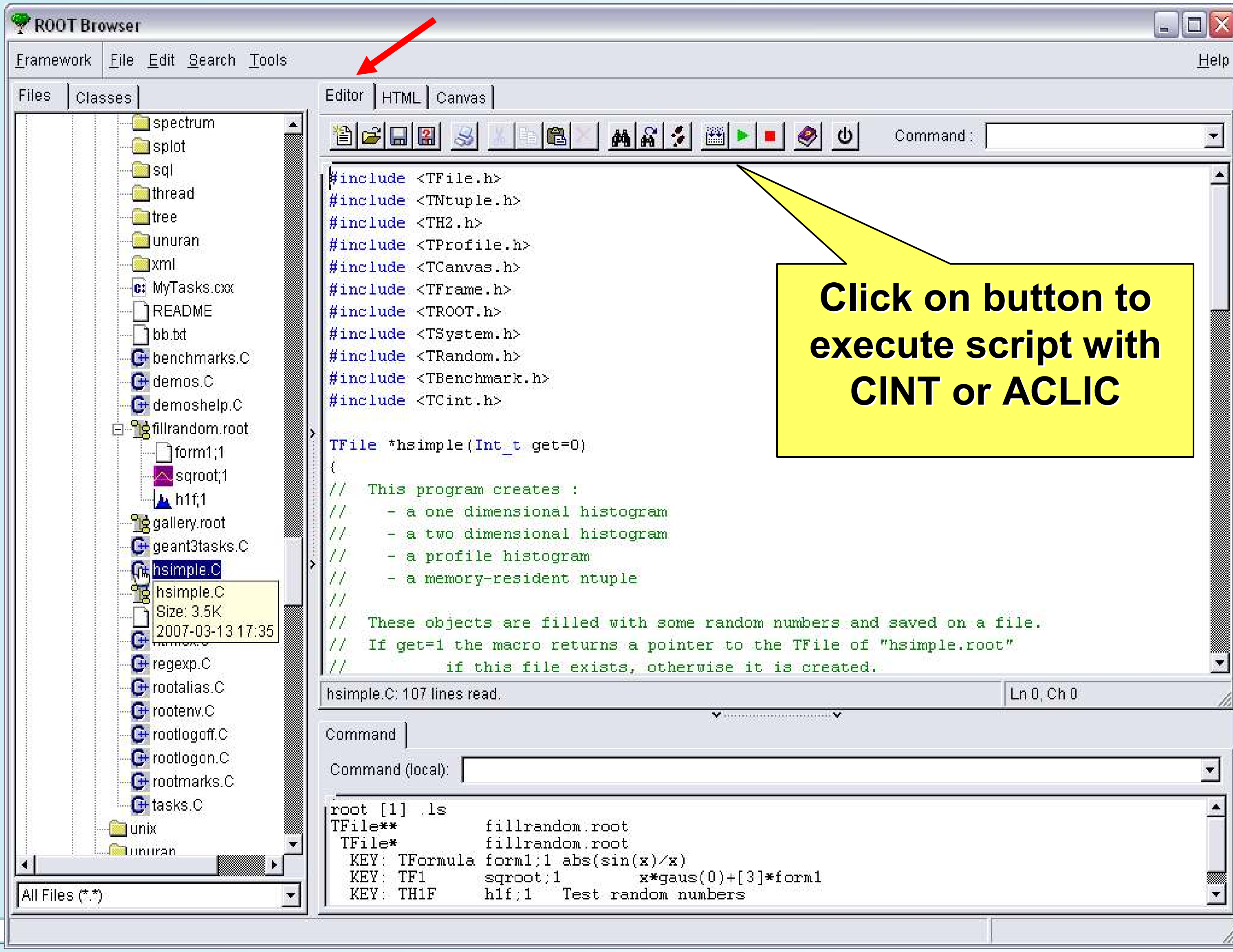
Hist Browser + stdin/stdout



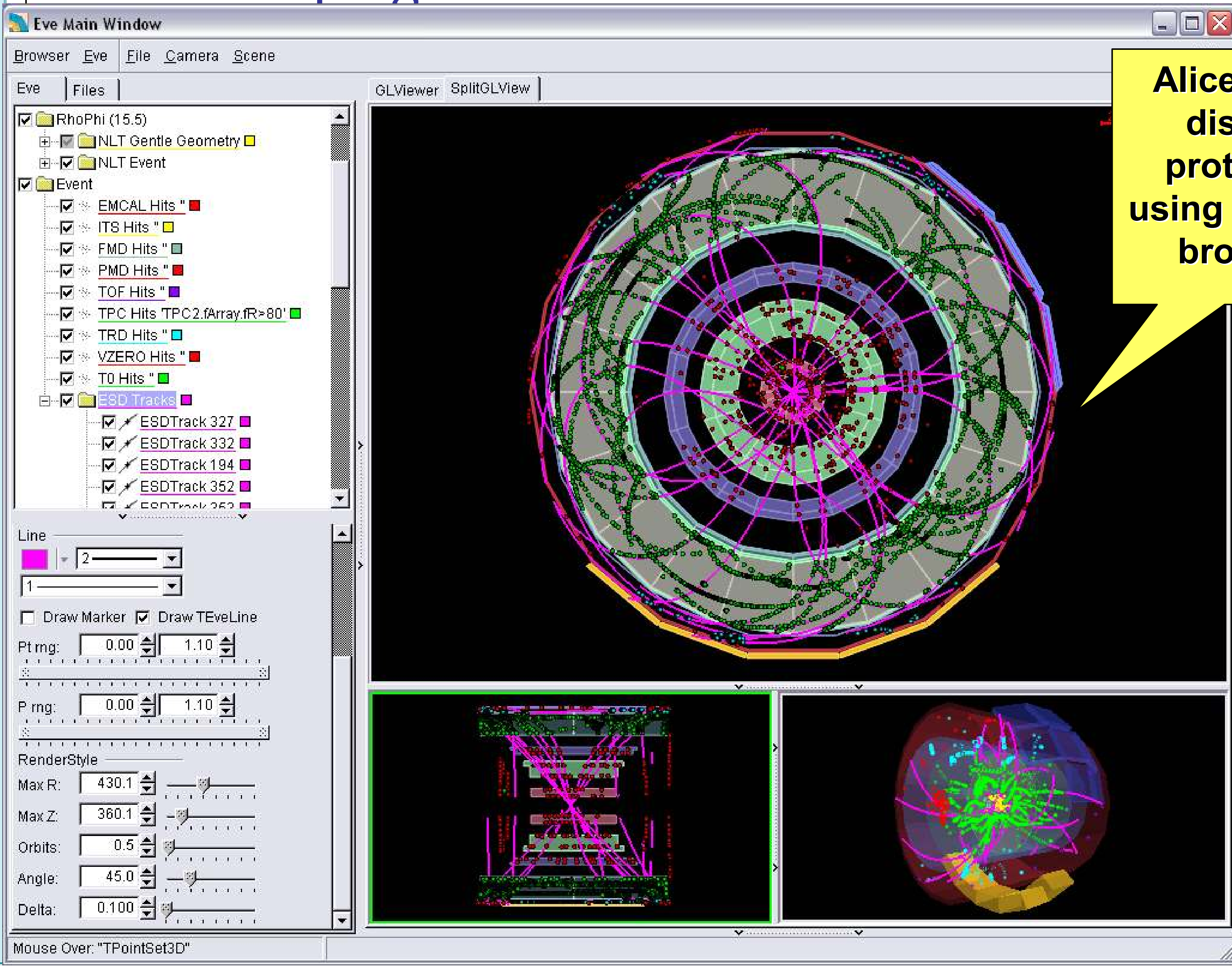
TGhtml web browser plug-in



Macro Manager/Editor plug-in



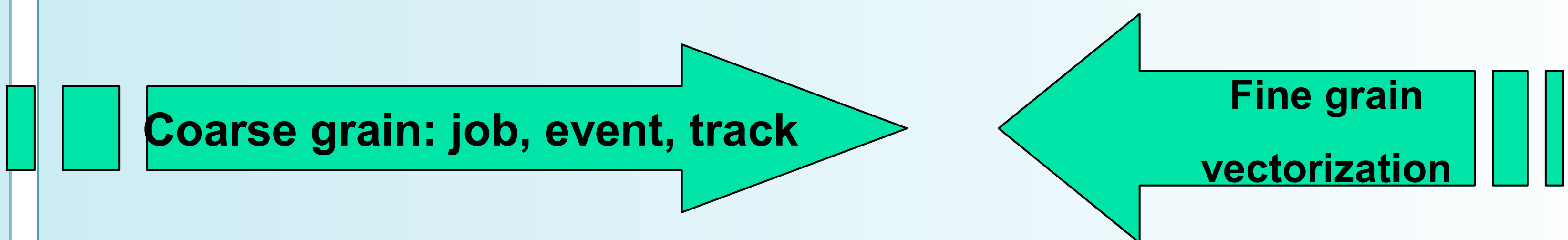
GL Viewer plug-in



Alice event display prototype using the new browser

Challenge ++
<u>Design for Parallelism</u>

- The GRID is a parallel engine. However it is unlikely that you will use the GRID software on your 32-core laptop.
- Restrict use of global variables and make tasks as independent as possible.
- Be thread-safe and (better) thread-aware
- Think Top->Down and Bottom->Up



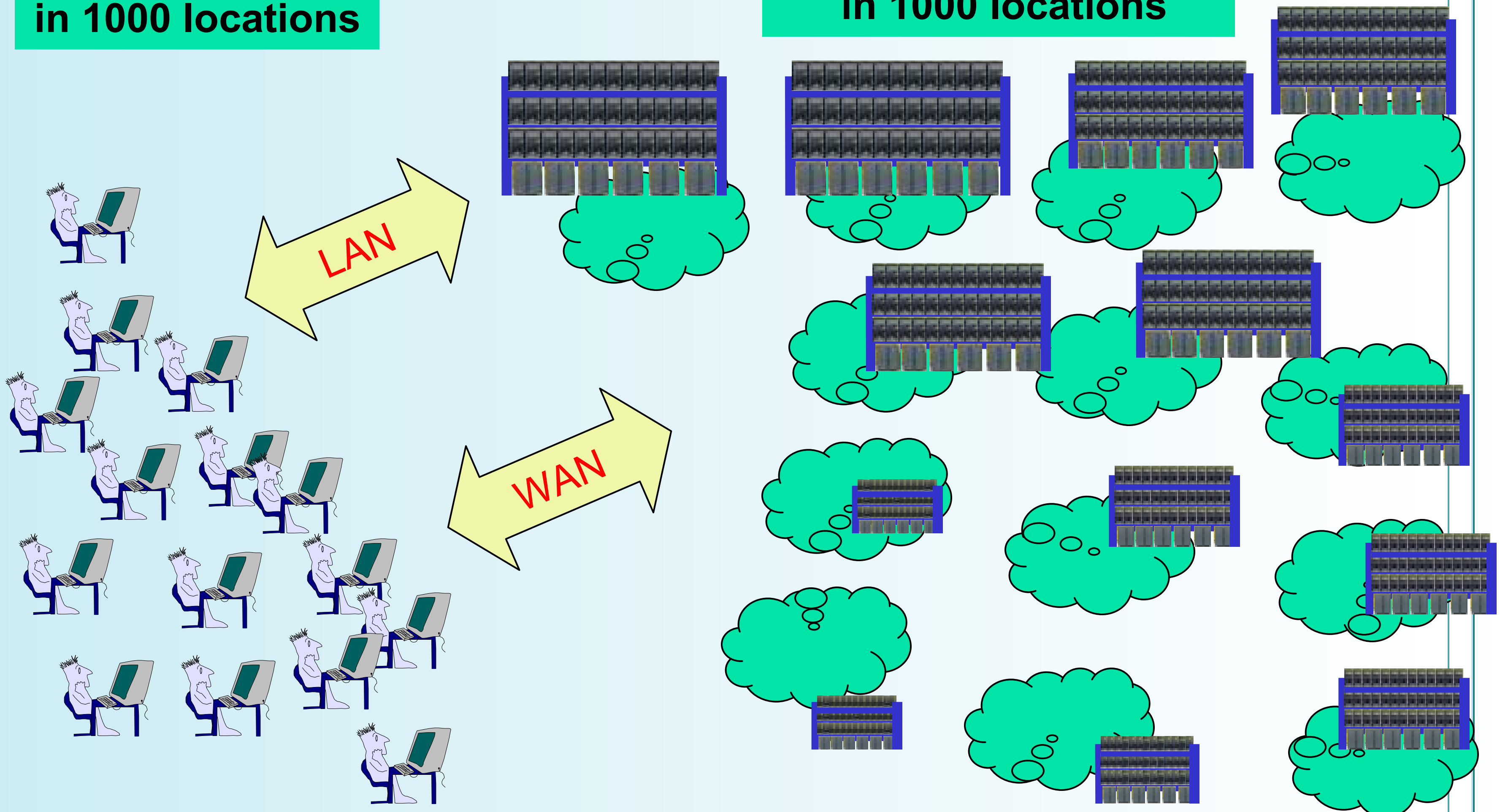
Towards a distributed system

5,000 physicists

in 1000 locations

100,000 computers

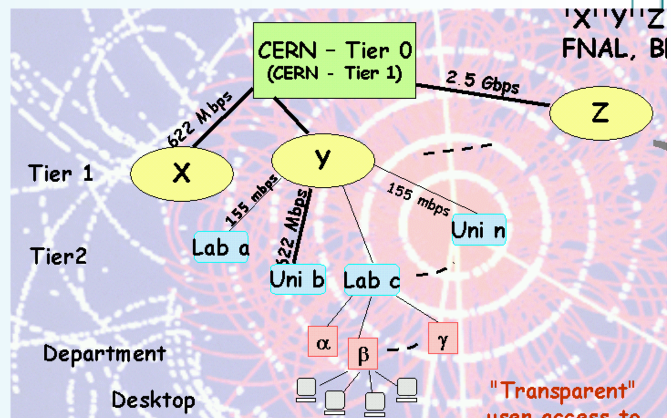
in 1000 locations



LHC collaborations (analysis steps)

Raw Data
(PetaBytes)

DAQ -> T0 -> T1

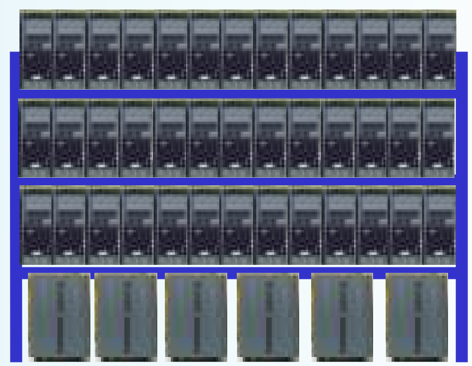


After reconstruction
(100 TeraBytes)

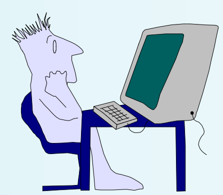
T1 -> T2

for analysis
(10 TeraBytes)

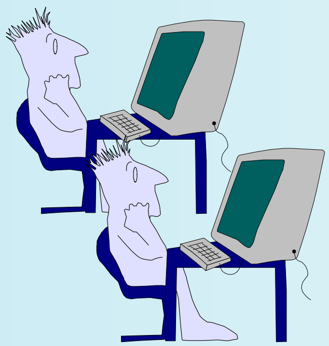
T2 -> T3



Analysis per physicist
(1 TeraByte)

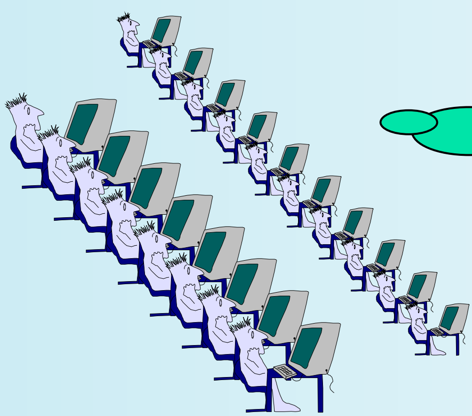


GRID: Users profile



Few big users submitting many long jobs (Monte Carlo, reconstruction)

They want to run many jobs in one month



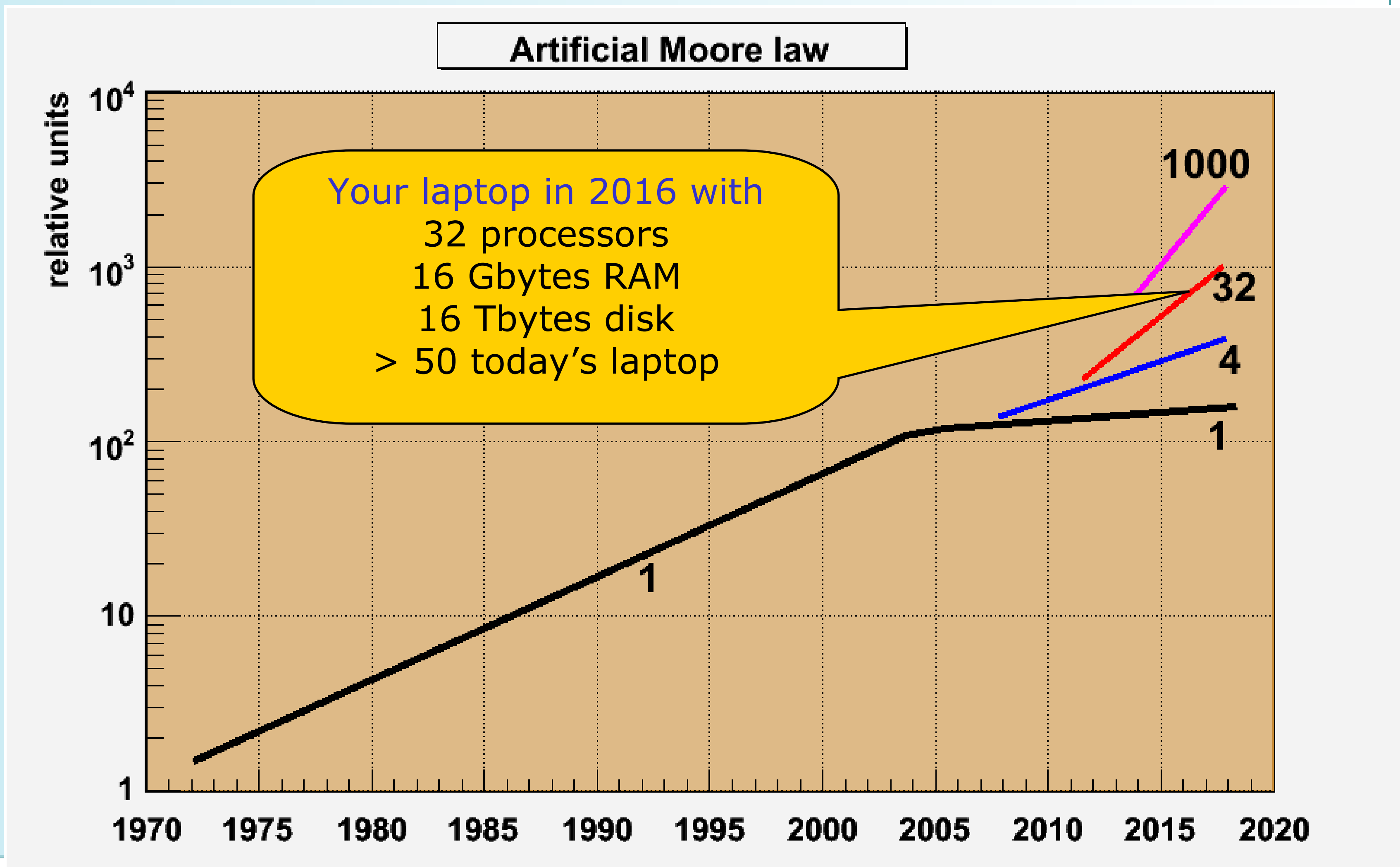
Many users submitting many short jobs (physics analysis)

They want to run many jobs in one hour or less

Many Use Cases

- Scenario 1: submit one batch job to the GRID. It runs somewhere with varying response times.
- Scenario 2: Use a splitter to submit many batch jobs to process many data sets (eg CRAB, Ganga, Alien). Output data sets are merged automatically. Success rate < 90%. You see the final results only when the last job has been received and all results merged.
- Scenario 3: Use PROOF (automatic splitter and merger). Success rate close to 100%. You can see intermediate feedback objects like histograms. You run from an interactive ROOT session.

Moore's law revisited



Challenge ++

Hardware will force parallelism

- Multi-Core (2-8)
- Many-Core (32-256)
- Mixture CPU + GPU-like (or FAT and MINI cores)
- Virtualization
- May be a new technology?
- Parallelism: a must

Challenge ++
Design for Client-Server

- The majority of today's applications are client-server (xrootd, Dcache, sql, etc).
- This trend will increase.
- Be able to stream objects or objects collections.
- Server logic robust against client changes.
- Server able to execute dynamic plug-ins.
- Must be robust against client or network crash

Challenge ++
LAN and WAN I/O caches

- Must be able to work very efficiently across fat pipes but with high latencies.
- Must be able to cache portions or full files on a local cache.
- This requires changes in data servers (Castor, Dcache, xrootd). These tools will have to **interoperate**.
- The ROOT file info must be given to these systems for optimum performance. See TTreeCache improvements.

Disk cache improvements with high latency networks

- The file is on a CERN machine connected to the CERN LAN at at 100MB/s.
- The client **A** is on the same machine as the file (local read)
- The client **F** is connected via ADSL with a bandwith of 8Mbits/s and a latency of 70 milliseconds (Mac Intel Coreduo 2Ghz).
- The client **G** is connected via a 10Gbits/s to a CERN machine via Caltech latency 240 ms.
- The times reported in the table are realtime seconds

One query to a 280 MB Tree I/O = 16.6 MB

client	latency(ms)	cachesize=0	cachesize=64KB	cachesize=10MB
A	0.0	3.4	3.4	3.4
F	72.0	743.7	48.3	28.0
G	240.0	>1800s	125.4s	9.9s

We expect to reach 4.5 s

Challenge ++
Executing Anywhere from Anywhere

- One should be able to start an application from any web browser.
- The local UI and GUI can execute transparently on a remote process.
- The resulting objects are streamed to the local session for fast visualization. (and not via an X11 server!)
- Prototype in latest ROOT using ssh technology.

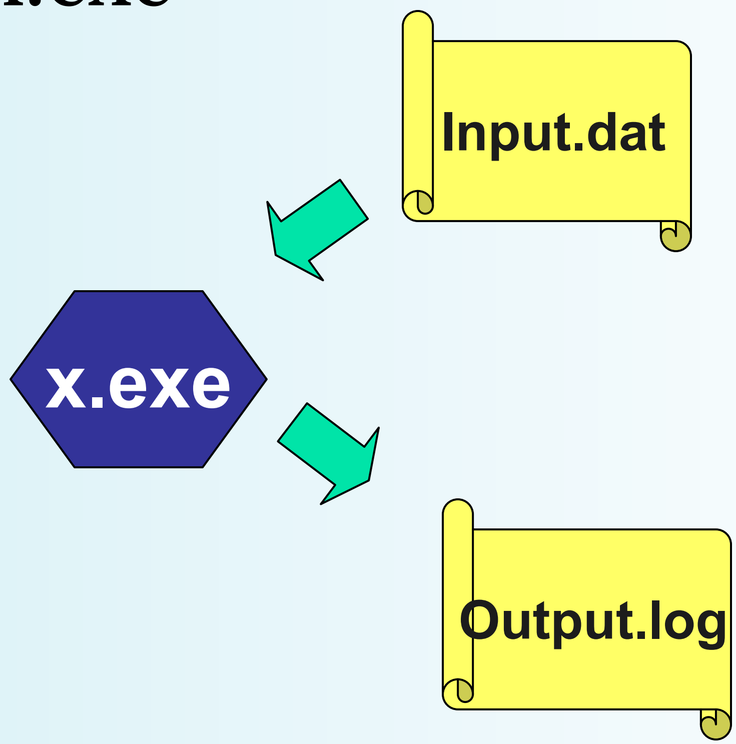
```
root > .R lxplus.cern.ch
lxplus > .x doSomething.C
lxplus > .R
root > //edit the local canvas
```

Challenge ++
Evolution of the Execution Model

- From stand alone modules
- To shared libs
- To plug-in managers
- To distributed computing
- To distributed and parallel computing

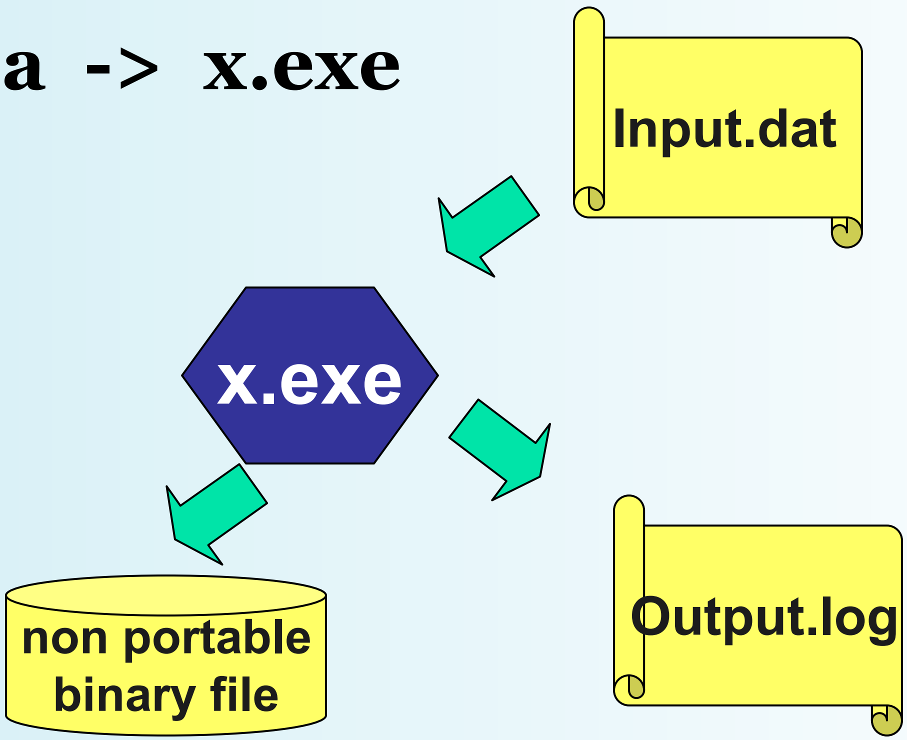
Executable module in 1968

- x.f -> x.o -> x.exe



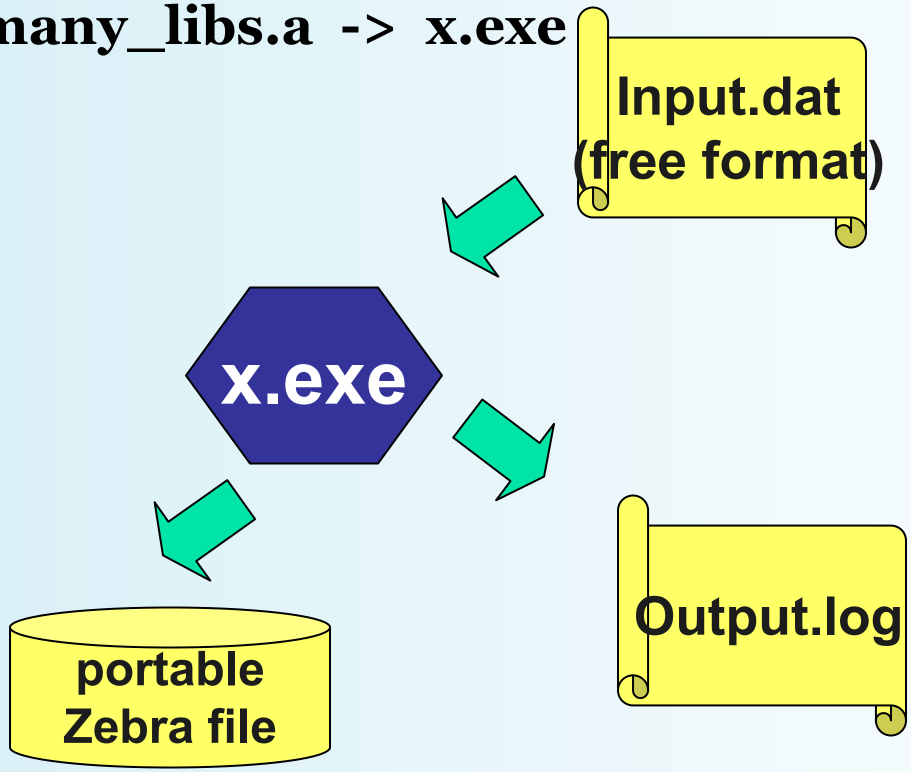
Executable module in 1978

- **x.f -> x.o**
- **x.o + libs.a -> x.exe**



Executable module in 1988

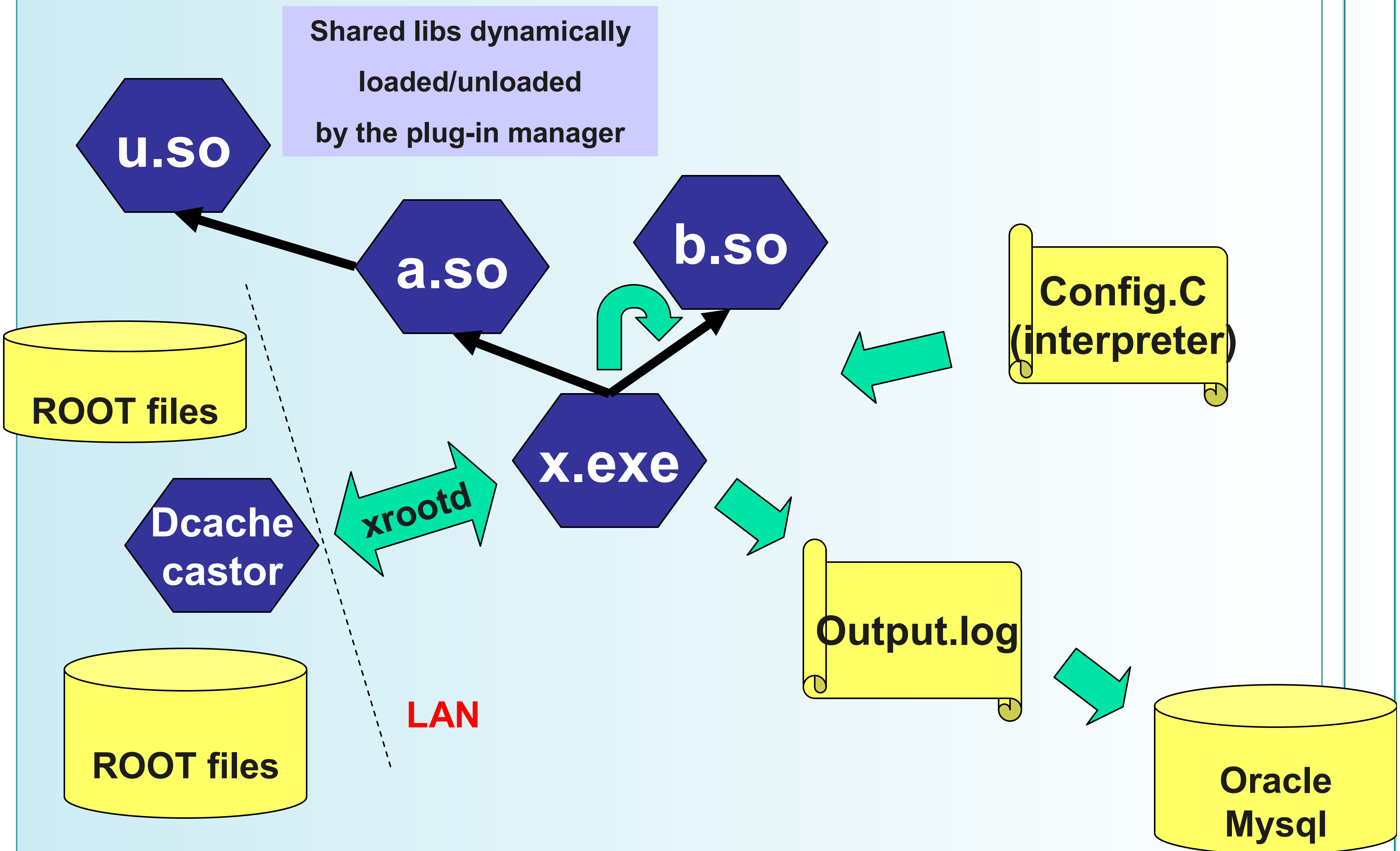
- **many_x.f -> many_x.o**
- **many_x.o + many_libs.a -> x.exe**



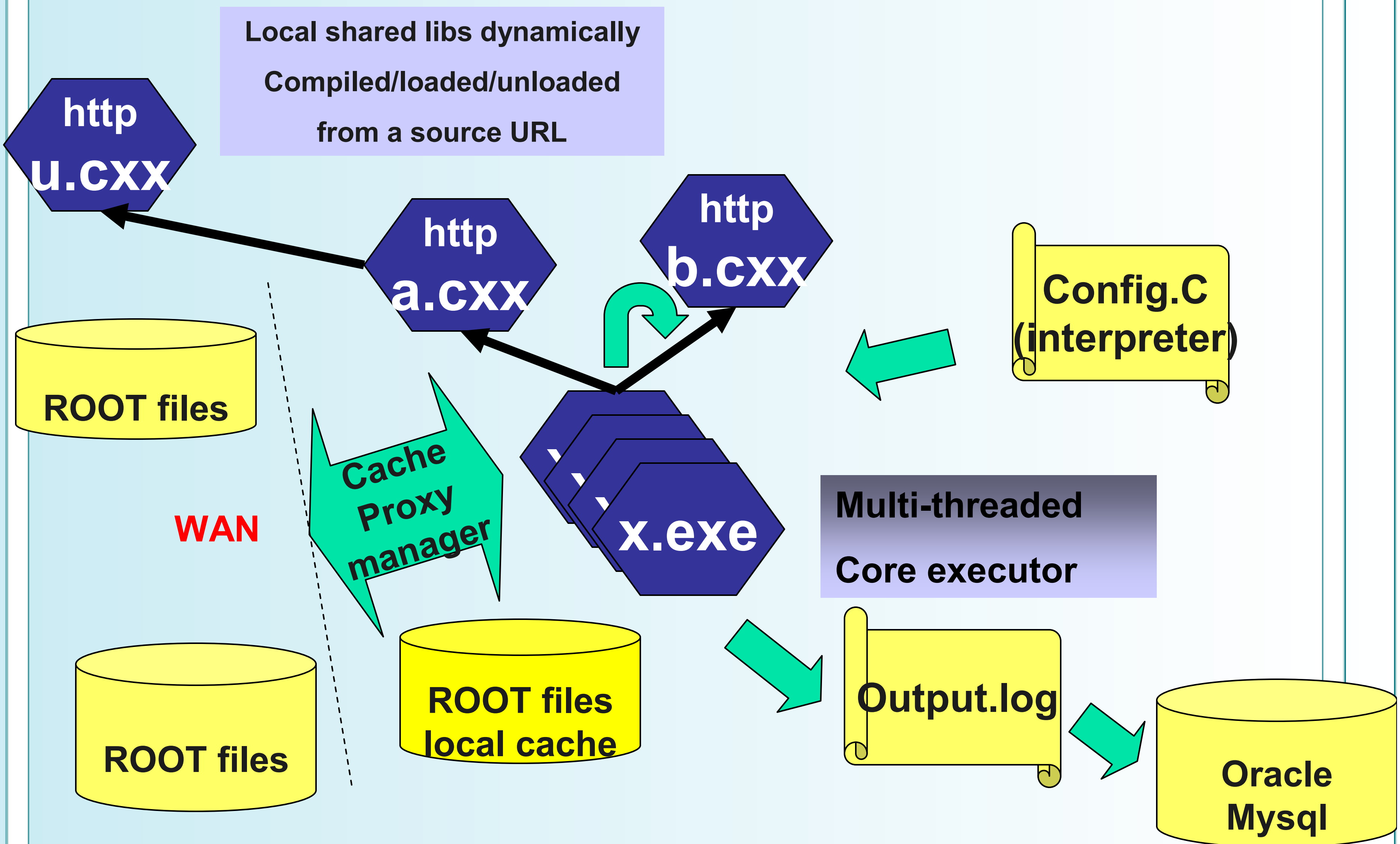
Executable module in 1998

- many_x.f -> many_x.o
- many_x.o + some_libs.a
- + many_libs.so -> x.exe

Executable module in 2008



Executable module in 2018 ?



Challenge ++
Software Development Tools

- better integration with Xcode, VisualStudio or like
- fast memory checkers
- faster valgrind
- faster profilers
- Better tools to debug parallel applications
- Code checkers and smell detection
- Better html page generators

Challenge ++
Distributed Code Management

- patchy, cmz -> cvs
- cvs -> svn
- cmt? scram? (managing dependencies)
- automatic project creation from cvs/svn to VisualStudio or Xcode and vice-versa

Challenge ++
Simplification of Software Distribution

- tar files
- source + make
- install from http://source
- install from http://binary proxy
- install on demand via plugin manager, autoloader
- automatic updates
- time to install
- fraction of code used

See BOOT
Project
First release
In 2008 ?

Conclusions

- Applications becoming more and more complex and distributed over the net, it is essential to:
  - Minimize interdepencies by providing a clean hierarchy of modular systems with robust components underneath.
  - Use as much as possible dynamic object managers (collections in files, browsers, tasks, folders, etc)
- The new hardware is pushing us to consider both fine grain and coarse grain parallelism
- More complexity must push us for:
  - Simpler and simpler user interfaces
  - Simpler software installation from sources on the web..

History of ROOT I/O

Streaming, Reflection, TFile,
Schema Evolution

ROOT I/O History

1994

- Version 0.9
  - Hand-written Streamers

1996

- Version 1
  - Streamers generated via rootcint
  - Support for Class Versions

1998

- Version 2.25
  - Support for ByteCount
  - Several attempts to introduce automatic class evolution
  - Simple support for STL
  - Only hand coded and generated streamer function, Schema evolution done by hand
  - I/O requires : ClassDef, ClassImp and CINT Dictionary

2000

2001

- Version 2.26 – 3.00
  - **Automatic schema evolution**
  - **Use TStreamerInfo (with info from dictionary) to drive a general I/O routine.**
  - **Self describing files**
  - **MakeProject** can regenerate the file's classes layout

ROOT I/O History

2002

- Version 3.03/05
  - **Lift need for** ClassDef and ClassImp for classes not inheriting from TObject
  - **Any non TObject class** can be saved inside a TTree or as part of a TObject-class
  - **TRef/TRefArray**

2004

- Version 4.00/08
  - Automatic versioning of 'Foreign' classes
  - Non TObject classes can be saved directly in TDirectory
- Version 4.04/02
  - Large TTrees, TRef autoload
  - TTree interface improvements, Double32 enhancements

2005

- Version 5.08/00
  - Fast TTree merging, Indexing of TChains, **Complete STL support.**

2006

- Version 5.12/00
  - Prefetching, TTreeCache
  - TRef autoderefencing

2007

- Version 5.16/00
  - Improved modularization (libRio)

2008

- Version 5.22/00
  - **Data Model Evolution** (brought to your courtesy of BNL/STAR/ATLAS)

Early Days

- The fundamental elements of I/O are present:
  - platform independence
  - compression
  - TFile/TDirectory layout and structure
  - TTree
- ***Dictionaries*** are already the corner-stone of the I/O
  - Allow streaming of user class with minimal intrusion and no complex ddl system.
- rootcint generated default C++ Streamer function

- Any schema evolution required to maintain the streamer functions by hand

Streamers in 0.90/08

```
class TAxis : public
TNamed,
public TAttAxis {

private:
  Int_t         fNbins;
  Axis_t        fXmin;
  Axis_t        fXmax;
  TArrayF       fXbins;
  Char_t       *fXlabels;
```

rootcint

```
void TAxis::Streamer(TBuffer &b)
{
   if (b.IsReading()) {
      Version_t v = b.ReadVersion();
      TNamed::Streamer(b);
      TAttAxis::Streamer(b);
      b >> fNbins;
      b >> fXmin;
      b >> fXmax;
      fXbins.Streamer(b);
   } else {
      b.WriteVersion(TAxis::IsA());
      TNamed::Streamer(b);
      TAttAxis::Streamer(b);
      b << fNbins;
      b << fXmin;
      b << fXmax;
      fXbins.Streamer(b);
   }
}
```

Streamers in 2.25 – Byte Count

```
class TAxis : public TNamed,
public TAttAxis {

private:
   Int_t        fNbins;
   Axis_t       fXmin;
   Axis_t       fXmax;
   TArrayF      fXbins;
   Char_t      *fXlabels;
   Int_t        fFirst;
   Int_t        fLast;
   TString      fTimeFormat;
   Bool_t       fTimeDisplay;
   TObject     *fParent;
```

rootcint

```
void TAxis::Streamer(TBuffer &R__b) {
   UInt_t R__s, R__c;
   if (R__b.IsReading()) {
      Version_t R__v = R__b.ReadVersion(&R__s, &R__c);
      TNamed::Streamer(R__b);
      TAttAxis::Streamer(R__b);
      R__b >> fNbins;
      R__b >> fXmin;
      R__b >> fXmax;
      fXbins.Streamer(R__b);
      R__b >> fFirst;
      R__b >> fLast;
      R__b >> fTimeDisplay;
      fTimeFormat.Streamer(R__b);
      R__b.CheckByteCount(R__s, R__c, TAxis::IsA());
   } else {
      R__c = R__b.WriteVersion(TAxis::IsA(), kTRUE);
      TNamed::Streamer(R__b);
      TAttAxis::Streamer(R__b);
      R__b << fNbins;
      R__b << fXmin;
      R__b << fXmax;
      fXbins.Streamer(R__b);
      R__b << fFirst;
      R__b << fLast;
      R__b << fTimeDisplay;
      fTimeFormat.Streamer(R__b);
      R__b.SetByteCount(R__c, kTRUE);
   }
}
```

Old Streamers in 2.25 – Schema Evolution

```
class TAxis : public TNamed,
public TAttAxis {

private:
   Int_t         fNbins;
   Axis_t        fXmin;
   Axis_t        fXmax;
   TArrayF       fXbins;
   Char_t       *fXlabels;
   Int_t         fFirst;
   Int_t         fLast;
   TString       fTimeFormat;
   Bool_t        fTimeDisplay;
   TObject      *fParent;
```

Developer

```
void TAxis::Streamer(TBuffer &R__b) {
   UInt_t R__s, R__c;
   if (R__b.IsReading()) {
      Version_t R__v = R__b.ReadVersion(&R__s, &R__c);
      TNamed::Streamer(R__b);
      TAttAxis::Streamer(R__b);
      R__b >> fNbins;
      R__b >> fXmin;
      R__b >> fXmax;
      fXbins.Streamer(R__b);
      if (R__v > 2) {
         R__b >> fFirst;
         R__b >> fLast;
      }
      if (R__v > 3) {
         R__b >> fTimeDisplay;
         fTimeFormat.Streamer(R__b);
      } else {
         SetTimeFormat();
      }
      R__b.CheckByteCount(R__s, R__c, TAxis::IsA());
   } else {
      R__c = R__b.WriteVersion(TAxis::IsA(), kTRUE);
      TNamed::Streamer(R__b);
      TAttAxis::Streamer(R__b);
      R__b << fNbins;
      R__b << fXmin;
      R__b << fXmax;
      fXbins.Streamer(R__b);
      R__b << fFirst;
      R__b << fLast;
      R__b << fTimeDisplay;
      fTimeFormat.Streamer(R__b);
      R__b.SetByteCount(R__c, kTRUE);
   }
}
```

2001 - StreamerInfo

- ROOT File are now self describing
  - Dictionary for persistent classes written to the file when closing the file.
  - ROOT files can be read by foreign readers (JAS for example)
  - Support for Backward and Forward compatibility
  - Files created in 2003 can be readable in 2015
  - Classes (data objects) for all objects in a file can be regenerated via TFile::MakeProject
  - Data can be read without the original code
- Provide for automatic schema evolution
  - Change the order of the members
  - Change simple data type (float to int)
  - Add or remove data members, base classes
  - Migrate a member to base class
- Basic support for STL container
  - does not support nested containers directly
  - can not ‘split’ STL containers
  - no schema evolution to and from different container types.

Streamers in 3.00 - StreamerInfo

```
class TAxis : public TNamed,
public TAttAxis {

private:
   Int_t         fNbins;
   Axis_t        fXmin;
   Axis_t        fXmax;
   TArrayF       fXbins;
   Char_t       *fXlabels;      //!
   Int_t         fFirst;
   Int_t         fLast;
   TString       fTimeFormat;
   Bool_t        fTimeDisplay;
   TObject      *fParent;       //!
```

```
void TAxis::Streamer(TBuffer &R__b)
{
   // Stream an object of class TAxis.

   if (R__b.IsReading()) {
      UInt_t R__s, R__c;
      Version_t R__v = R__b.ReadVersion(&R__s, &R__c);
      if (R__v > 5) {
         TAxis::Class()->ReadBuffer(R__b, this, R__v, R__s, R__c);
         return;
      }
      //====process old versions before automatic schema evolution
      ....
      //====end of old versions

   } else {
      TAxis::Class()->WriteBuffer(R__b,this);
   }
}
```

developer

Seeing classes in a file

```
Root > f.ShowStreamerInfo()
StreamerInfo for class: ATLFMuon, version=1
  BASE          TObject         offset=  0 type=66 Basic ROOT object
  BASE          TAtt3D          offset=  0 type= 0 3D attributes
  Int_t         m_KFcode        offset=  0 type= 3 Muon KF-code
  Int_t         m_MCParticle    offset=  0 type= 3 Muon position in MCParticles list
  Int_t         m_KFmother      offset=  0 type= 3 Muon mother KF-code
  Int_t         m_UseFlag       offset=  0 type= 3 Muon energy usage flag (0 for used in clusters)
  Int_t         m_Isolated      offset=  0 type= 3 Muon isolation (1 for isolated)
  Float_t       m_Eta           offset=  0 type= 5 Eta coordinate
  Float_t       m_Phi           offset=  0 type= 5 Phi coordinate
  Float_t       m_PT            offset=  0 type= 5 Transverse energy
  Int_t         m_Trigger       offset=  0 type= 3 Result of trigger

StreamerInfo for class: ATLFElectron, version=1
  BASE          TObject         offset=  0 type=66 Basic ROOT object
  BASE          TAtt3D          offset=  0 type= 0 3D attributes
  Int_t         m_KFcode        offset=  0 type= 3 Electron KF-code
  Int_t         m_MCParticle    offset=  0 type= 3 Electron position in MCParticles list
  Int_t         m_KFmother      offset=  0 type= 3 Electron mother KF-code
  Float_t       m_Eta           offset=  0 type= 5 Eta coordinate
  Float_t       m_Phi           offset=  0 type= 5 Phi coordinate
  Float_t       m_PT            offset=  0 type= 5 Transverse energy

StreamerInfo for class: ATLFPhoton, version=1
  BASE          TObject         offset=  0 type=66 Basic ROOT object
  BASE          TAtt3D          offset=  0 type= 0 3D attributes
  Int_t         m_KFcode        offset=  0 type= 3 Photon KF-code
  Int_t         m_MCParticle    offset=  0 type= 3 Photon position in MCParticles list
  Int_t         m_KFmother      offset=  0 type= 3 Photon mother KF-code
  Float_t       m_Eta           offset=  0 type= 5 Eta coordinate
  Float_t       m_Phi           offset=  0 type= 5 Phi coordinate
  Float_t       m_PT            offset=  0 type= 5 Transverse energy

StreamerInfo for class: ATLFJet, version=1
  BASE          TObject         offset=  0 type=66 Basic ROOT object
  BASE          TAtt3D          offset=  0 type= 0 3D attributes
  Int_t         m_KFcode        offset=  0 type= 3 Jet KF-code
  Int_t         m_Ncells        offset=  0 type= 3 Number of cells used for reconstruction
  Int_t         m_Nparticles    offset=  0 type= 3 Number of particles assigned to jet
  Int_t         m_Part          offset=  0 type= 3 Position in MCParticle list of matching b-quark/c-qu
  Float_t       m_Eta0          offset=  0 type= 5 Eta position of initiator cell
  Float_t       m_Phi0          offset=  0 type= 5 Phi position of initiator cell
  Float_t       m_Eta           offset=  0 type= 5 Eta  of jet bary-center
  Float_t       m_Phi           offset=  0 type= 5 Phi  of jet bary-center
  Float_t       m_PT            offset=  0 type= 5 Transverse momentum of jet

....
```

2001 - examples

```
enum {kSize=10};
   char            fType[20];      //array of 20 chars
   Int_t           fNtrack;        //number of tracks
   Int_t           fNvertex;       //number of vertices
   Int_t           fX[kSize];      //an array where dimension is an enum
   UInt_t          fFlag;          //bit pattern event flag
   Float_t         fMatrix[4][4];  //a two-dim array
   Float_t        *fDistance;      //[fNvertex] array of floats of length fNvertex
   Double_t        fTemperature;   //event temperature
   TString        *fTstringp;      //[fNvertex] array of TString
   TString         fNames[12];     //array of TString
   TAxis           fXaxis;         //example of class derived from TObject
   TAxis           fYaxis[3];      //array of objects
   TAxis          *fVaxis[3];      //pointer to an array of TAxis
   TAxis          *fPaxis;         //[fNvertex] array of TAxis of length fNvertex
   TAxis         **fQaxis;         //[fNvertex] array of pointers to TAxis objects
   TDatime         fDatime;        //date and time
   EventHeader     fEvtHdr;        //example of class not derived from TObject
   TObjArray       fObjArray;      //An object array of TObject*
   TClonesArray   *fTracks;        //-> array of tracks
   TH1F           *fH;             //-> pointer to an histogram
   TArrayF         fArrayF;        //an array of floats
   TArrayI        *fArrayI;        //a pointer to an array of integers
………………..(see next)
```

2001- Support for STL

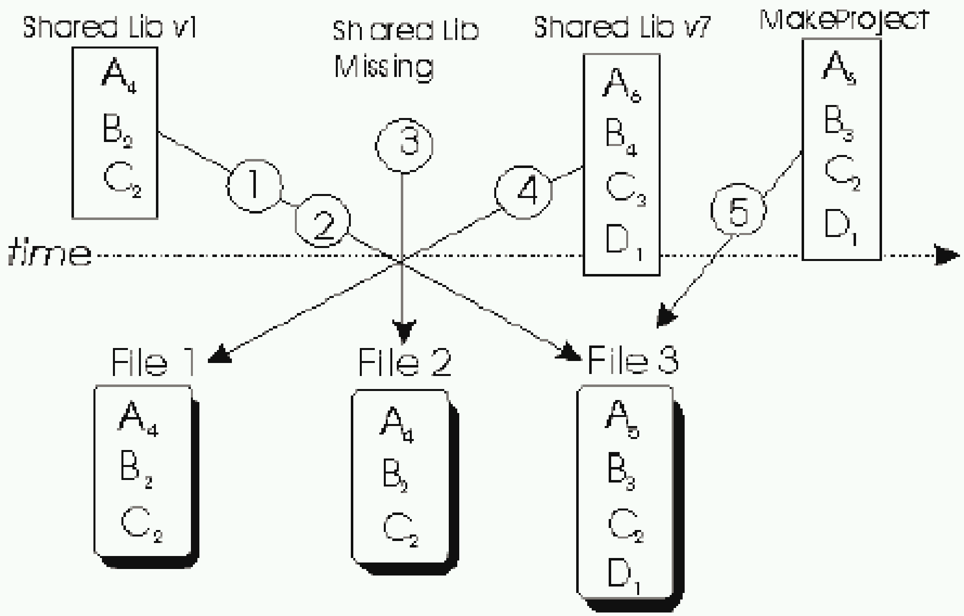
```
vector<int>             fVectorint;       //STL vector on ints
vector<short>           fVectorshort;     //STL vector of shorts
vector<double>          fVectorD[4];      //array of STL vectors of doubles
vector<TLine>           fVectorTLine;     //|| STL vector of TLine objects
vector<TObject>        *fVectorTobject;   //|| pointer to an STL vector
vector<TNamed>         *fVectorTnamed[6]; //|| array of pointers to STL vectors
deque<TAttLine>         fDeque;           //STL deque
list<const TObject*>    fVectorTobjectp;  //STL list of pointers to objects
list<string>           *fListString;      //STL list of strings
list<string *>          fListStringp;     //STL list of pointers to strings
map<TNamed*,int>        fMapTNamedp;      //STL map

map<TString,TList*>     fMapList;         //STL map
map<TAxis*,int>        *fMapTAxisp;       //pointer to STL map
set<TAxis*>             fSetTAxis;        //STL set
set<TAxis*>            *fSetTAxisp;       //pointer to STL set
multimap<TNamed*,int>   fMultiMapTNamedp; //STL multimap
multiset<TAxis*>       *fMultiSetTAxisp;  //pointer to STL multiset
string                  fString;          //C++ standard string
string                 *fStringp;         //pointer to standard C++ string
UShortVector            fUshort;          //class with an STL vector as base class
```

Need custom Streamer for these complex cases

```
vector<vector<TAxis *> >  fVectAxis;    //!STL vector of vectors of TAxis*
map<string,vector<int> >  fMapString;   //!STL map of string/vector
deque<pair<float,float> > fDequePair;   //!STL deque of pair
```

Automatic Schema Evolution



1) An old version of a shared library and a file with new class definitions. This can be the case when someone has not updated the library and is reading a new file.

2) Reading a file with a shared library that is missing a class definition (i.e. missing class D).

3) Reading a file without any class definitions. This can be the case where the class definition is lost, or unavailable.

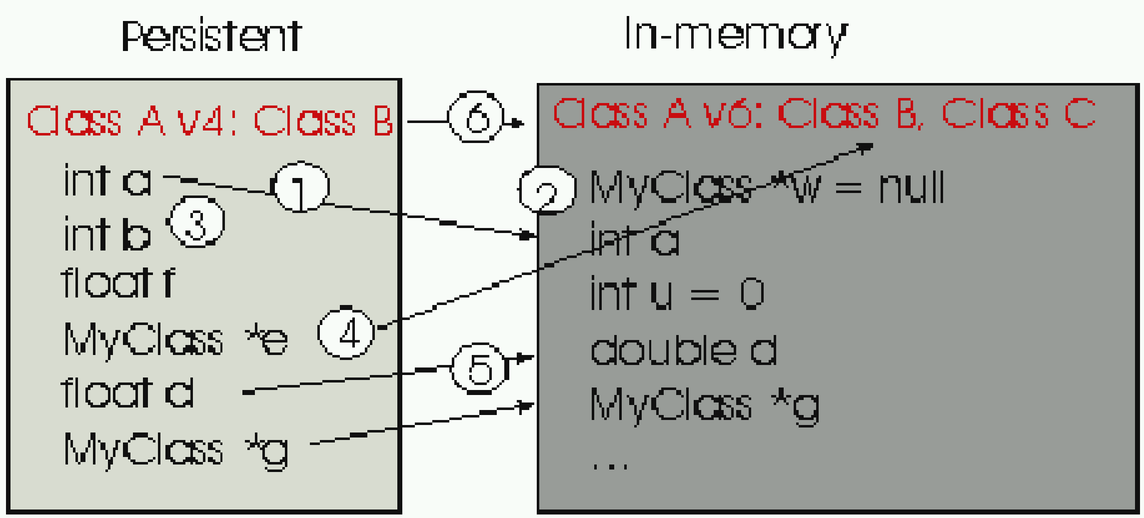
4) The current version of a shared library and an old file with old class versions (backward compatibility). This is often the case when reading old data.

5) Reading a file with a shared library built with `MakeProject`. This is the case when someone has already read the data without a shared library and has used ROOT's `MakeProject` feature to reconstruct the class definitions and shared library (`MakeProject` is explained in detail later on).

Auto Schema Evolution (2)

In case of a mismatch between the in-memory version and the persistent version of a class, ROOT maps the persistent one to the one in memory. This allows you to change the class definition at will, for example:

1) Change the order of data members in the class.
2) Add new data members. By default the value of the missing member will be 0 or in case of an object it will be set to null.
3) Remove data members.
4) Move a data member to a base class or vice –versa.
5) Change the type of a member if it is a simple type or a pointer to a simple type. If a loss of precision occurs, a warning is given.
6) Add or remove a base class



TFile::MakeProject

```
////////////////////////////////////////////////////////////
//   This class has been generated by TFile::MakeProject
//     (Mon May 28 19:34:37 2001 by ROOT version 3.01/03)
//      from the StreamerInfo in file atlfast.root
////////////////////////////////////////////////////////////


#ifndef ATLFElectron_h
#define ATLFElectron_h

#include "TObject.h"
#include "TAtt3D.h"

class ATLFElectron : public TObject , public TAtt3D {

public:
   Int_t        m_KFcode;       //Electron KF-code
   Int_t        m_MCParticle;   //Electron position in MCParticles list
   Int_t        m_KFmother;     //Electron mother KF-code
   Float_t      m_Eta;          //Eta coordinate
   Float_t      m_Phi;          //Phi coordinate
   Float_t      m_PT;           //Transverse energy

   ATLFElectron() {;}
   virtual ~ATLFElectron() {;}

   ClassDef(ATLFElectron,1) //
};

   ClassImp(ATLFElectron)
#endif
```

All necessary header files are included

Comments preserved

Can do I/O Inspect Browse,etc

2002 – I/O for Non-TObject

- Saving non-instrumented Classes
  - Being able to save in a ROOT file objects from library that you can NOT modify at all.
  - Being able to easily save objects that do not inherit from TObject.

- Lift limitation on number of template parameters

2005 - Generalized support for collections

- Abstract Interface (TVirtualCollectionProxy)
  - Initial Prototype and fundamental Concepts by Victor Perevoztchikov (BNL)
  - Can be implemented for almost any collections
  - Allows
    - Splitting (when possible)
    - Use in Tree Query (with automatic looping)
    - Member-wise streaming (as opposed to Object wise streaming)
  - Also
    - Arbitrary nesting of STL containers
    - Reading of STL containers without original code (Emulated mode)
  - Extended in 2008 to also support splitting of container of pointers.

TRef/TRefArray

- 2002: Allow for reference that span across branches or keys.
  - Designed as light weight entities
  - Assume large number of TRefs per event
  - Very fast dereferencing (direct access tables)
  - Not designed for finding an object in a different file
  - Occupies in average 2.5 bytes in the file

- 2004: Reference Autoload
  - TTree can be set to allow for automatic loading of the branch containing the referenced object

- 2006: Reference Autoderefencing
  - TTree::Draw can transparently drill through TRefs (*skipping complex call to GetObject and casting*)
  - Autoderefencing system flexible enough to support ***any*** reference type.

And Some More

- Improved Modularization (2007)
  - libCore, libRIO, libTTree, libTTreePlayer
- Improved compression tunning (2004,2005,2007)
  - Double32, Float16, saved in as few bits as requested.
- FastMerging (2005)
  - Improve performance of concatenation jobs by skipping uncompressing (zip) and unstreaming (object creation) steps [Pioneered by CDF]
- Extension of the output format (2004)
  - XML
  - Relational Database
- TFileStager / TTreeCache (2008)
  - Improve performance over slow link or low latency links
- Autodetection of user types in TTree interface (2007/8)
- Unzipping of basket in background (2008)

Data Model Evolution

- Limitation of Automatic schema evolution
  - Handle only removal, addition of members and change in simple type
  - Does not support change in complex type, change in semantic (like units)
- Limitation of hand written schema evolution
  - Since it requires a streamer function it can not be used in split mode

- Data Model Evolution solves this issues
  (brought to your courtesy of BNL/STAR/ATLAS)
- Capabilities:
  - Assign values to transient data members
  - Rename classes
  - Rename data members
  - Change the shape of the data structures or convert one class structure to another
  - Change the meaning of data members
  - Can access the TBuffer directly if needed
  - Ensure that the objects in collections are handled in the same way as the ones stored separately
  - Make things operational also in bare ROOT mode
  - ***Supported in object-wise, member-wise and split modes.***

Data Model Evolution

- Setting a transient member

```
#pragma read sourceClass="ACache" targetClass="ACache" \
   source="" version="[1-]" target="zcalc" \
   code="{ zcalc = false; }"
```

- Setting a new member from 2 removed members

```
#pragma read sourceClass="ACache" targetClass="ACache" \
   source="int x; int y;" version="[8]" target="z" \
   code="{ z = onfile.x*1000 + onfile.y*10; }"
```

- Renaming a class

```
#pragma read sourceClass="ACache" targetClass="Axis" \
   source="int x; int y;"  version="[8]" target="z" \
   code="{ z = onfile.x*1000 + onfile.y*10; }"
#pragma read sourceClass="ACache" version="[9]" targetClass="Axis";
```